

Министерство образования и науки Украины
Донбасская государственная машиностроительная академия

А.Ю. Мельников

**ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ АНАЛИЗ
И ПРОЕКТИРОВАНИЕ
ИНФОРМАЦИОННЫХ СИСТЕМ**

УЧЕБНОЕ ПОСОБИЕ

для студентов специальностей
«Экономическая кибернетика»
и
«Интеллектуальные системы принятия решений»

Утверждено на заседании
ученого совета ДГМА
Протокол № 1 от **28.09.2006г.**

Краматорск 2006

Рецензенты:

- В.Ф.СЫТНИК, доктор экономических наук, профессор, заведующий кафедрой Киевского национального экономического университета
- Р.Н.ЛЕПА, кандидат экономических наук, доцент, заведующий отделом Научно-исследовательского центра информационных технологий Института экономики промышленности Национальной академии наук Украины

Навчальний посібник містить основні теоретичні відомості по об'єктно-орієнтованому аналізу і проектуванню систем на уніфікованій мові моделювання UML, порядок роботи у середовищі IBM Rational Rose і приклади створення моделей систем в цьому середовищі.

Рекомендується студентам спеціальностей 7.050102 «Економічна кібернетика» і 7.080404 «Інтелектуальні системи прийняття рішень» як навчальний посібник при вивченні відповідних модулів курсів «Об'єктно-орієнтоване програмування», «Проектування інформаційних систем», «Системний аналіз і проектування систем обробки інформації» і «Технологія програмування і створення програмних продуктів», а також як довідник під час курсового і дипломного проектування.

Мельников А. Ю.

М48

Объектно-ориентированный анализ и проектирование информационных систем: Учебное пособие для студентов специальностей «Экономическая кибернетика» и «Интеллектуальные системы принятия решений». – Краматорск: ДГМА, 2006. – 184 с.

ISBN 966-379-103-9

Учебное пособие включает основные теоретические сведения по объектно-ориентированному анализу и проектированию систем на универсальном языке моделирования UML, порядок работы в среде IBM Rational Rose и примеры создания моделей систем в этой среде.

Рекомендуется студентам специальностей 7.050102 «Экономическая кибернетика» и 7.080404 «Интеллектуальные системы принятия решений» в качестве учебного пособия при изучении соответствующих модулей курсов «Объектно-ориентированное программирование», «Проектирование информационных систем», «Системный анализ и проектирование систем обработки информации» и «Технология программирования и создания программных продуктов», а также как справочное пособие в ходе курсового и дипломного проектирования.

УДК 004:681
ББК 32.973-01

ISBN 966-379-103-9

© Мельников А.Ю., 2006
© ДГМА, 2006

СОДЕРЖАНИЕ

Введение.....	5
1 Основные понятия объектно-ориентированного подхода.....	6
1.1 Объектная модель.....	7
1.2 Классы и объекты.....	10
1.3 Классификация.....	14
2 Унифицированный язык моделирования UML как средство проектирования программных систем и бизнес-процессов.....	17
2.1 Предыстория, основы и структура UML.....	18
2.2 Диаграмма концептуального моделирования – диаграмма вариантов использования (use case diagram).....	23
2.3 Диаграммы логического моделирования.....	31
2.3.1 Диаграмма классов (class diagram).....	31
2.3.2 Диаграмма кооперации (collaboration diagram).....	40
2.3.3 Диаграмма последовательности (sequence diagram).....	46
2.3.4 Диаграмма состояний (statechart diagram).....	50
2.3.5 Диаграмма деятельности (activity diagram).....	55
2.4 Диаграммы физического моделирования.....	62
2.4.1 Диаграмма компонентов (component diagram).....	62
2.4.2 Диаграмма развертывания (deployment diagram).....	65
3 Проектирование программных систем с использованием CASE-средства IBM Rational Rose.....	68
3.1 Общая характеристика инструментария IBM Rational Rose....	69
3.2 Пример разработки модели информационной системы в среде IBM Rational Rose.....	71
3.3 Генерация кода спроектированной модели в среде программирования.....	100
4 Примеры проектирования информационных систем.....	112
4.1 Информационная система для функционирования кадрового	

агентства.....	113
4.2 Информационная система для автоматизированного составления расписания занятий в высшем учебном заведении...	119
4.3 Информационная система для специализированного торгового предприятия.....	128
4.4 Информационная система для небольшой страховой компании.....	136
4.5 Информационная система для обеспечения функционирования финансового отдела предприятия.....	146
4.6 Информационная система для расчета себестоимости металлопродукции.....	152
4.7 Информационная система для учета и контроля готовой продукции.....	162
4.8 Информационная система для маркетинговых исследований и анализа надежности.....	170
Список литературы.....	180

ВВЕДЕНИЕ

Сложность современного программного обеспечения и высокая стоимость его разработки и сопровождения привели к возникновению нового – объектно-ориентированного – подхода к созданию информационных систем. При этом из-за участия в разработке этих систем десятков и сотен различных специалистов возникла необходимость построения предварительной модели системы до начала написания соответствующего программного кода. Основное требование к такой модели – быть понятной как заказчику программной системы, так и всем специалистам-разработчикам (системным аналитикам, программистам и т.п.).

В настоящее время в мире распространены три нотации визуального моделирования: IDEF (Icam DEFinition), ARIS (Architecture of Integrated Information Systems) и UML (Unified Modelling Language). Первые две применяются чаще всего при моделировании бизнес-процессов. Рассмотрению третьего, ставшего, по сути, мировым стандартом разработки программного обеспечения, и посвящено данное учебное пособие.

Пособие состоит из четырех частей – двух разделов теоретической направленности и двух – практической. Сначала в конспективной форме излагаются концепции объектно-ориентированного подхода к анализу и проектированию – понятия и определения объектной модели, классов и объектов и их возможных классификаций (за более подробной информацией следует обращаться к «классическому» труду Гради Буча [1]). Затем подробно описывается язык проектирования программных систем UML (в качестве базового учебника принят авторский труд Александра Леоненкова [2], который, в отличие от ряда переводных изданий [3-7], максимально удобно структурирован и предполагает путь «от простого к сложному»).

Третий раздел посвящен описанию CASE-средства проектирования программных систем «IBM Rational Rose», четвертый содержит примеры спроектированных систем (все представленные модели разработаны студентами специальности «Экономическая кибернетика» в рамках курсового и дипломного проектирования под руководством или с участием автора пособия).

1 ОСНОВНЫЕ ПОНЯТИЯ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПОДХОДА

Все методы проектирования программных систем можно объединить в три группы:

- структурное проектирование сверху вниз, в основе которого лежит алгоритмическая декомпозиция, и которое не может обеспечить создание сложных систем и неэффективно в объектно-ориентированных языках;
- метод потоков данных, когда система рассматривается как преобразователь входных потоков в выходные;
- объектно-ориентированное проектирование.

Объектно-ориентированное программирование (ООР – Object-oriented programming) – это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования. Следует различать «объектные» («объектно-базированные») и «объектно-ориентированные» языки программирования.

Объектно-ориентированное проектирование (OOD – Object-oriented design) – это методология проектирования, соединяющая в себе процесс объектной декомпозиции и приемы представления логической и физической, а также статической и динамической моделей проектируемой системы.

Объектно-ориентированный анализ (ООА – Object-oriented analysis) – это методология, при которой требования к системе воспринимаются с точки зрения классов и объектов, выявленных в предметной области.

В результате объектно-ориентированного анализа формируются модели, на которых основывается объектно-ориентированное проектирование, создающее фундамент для окончательной реализации системы с использованием методологии объектно-ориентированного программирования. Все три методологии базируются на понятиях **объектной модели**.

1.1 Объектная модель

Объектная модель является концептуальной базой объектно-ориентированного стиля. Она состоит из четырех главных элементов (абстрагирование, инкапсуляция, модульность, иерархия) и трех дополнительных (типизация, параллелизм, сохраняемость), которые полезны, но не обязательны.

Абстрагирование выделяет существенные характеристики некоторого объекта, отличающие его от всех других видов объектов, и, таким образом, четко определяет его концептуальные границы с точки зрения наблюдателя. Главный принцип абстрагирования – принцип наименьшего удивления, согласно которому абстракция должна охватывать все поведение объекта, но не приносить сюрпризов или побочных эффектов, лежащих вне ее сферы применимости.

Выбор правильного набора абстракций – самая главная задача.

Абстракции можно объединить в 4 группы.

Абстракция сущности: объект представляет собой полезную модель некой сущности в предметной области.

Абстракция поведения: объект состоит из обобщенного множества операций.

Абстракция виртуальной машины: объект группирует операции, которые либо вместе используются более высоким уровнем управления, либо сами используют некоторый набор операций более низкого уровня.

Произвольная абстракция: объект включает в себя набор операций, не имеющих друг с другом ничего общего.

Примеры абстракций в гидропонном хозяйстве: теплица, температура, влажность, освещение, кислотность, датчики, культуры, план выращивания.

Инкапсуляция – это процесс отделения друг от друга элементов объекта, определяющих его устройство и поведение; служит для того, чтобы изолировать контрактные обязательства абстракции от их реализации.

Абстрагирование и инкапсуляция дополняют друг друга: первое направлено на наблюдаемое поведение объекта, а вторая занимается его внутренним устройством. *Интерфейс* отражает внешнее поведение объ-

екта, описывая абстракцию поведения всех объектов данного класса; внутренняя *реализация* описывает представление этой абстракции и механизмы достижения желаемого поведения объекта.

(Следует обратить внимание: «Инкапсуляция не спасает от глупости; она защищает от ошибок, но не от жульничества» – Б. Страуструп).

Пример: простой нагреватель имеет расположение (конструктор), включатель, выключатель и проверку состояния. Усложненный нагреватель может быть связан с датчиком температуры и планом выращивания.

Модульность – это свойство системы разлагаться на внутренние связанные, но слабо связанные между собой модули. Логическое продолжение инкапсуляции, ее «практическая проекция».

Иерархия – это упорядочение абстракций, расположение их по уровням. Различают два вида иерархических структур: структура классов («is-a») и структура объектов («part of»).

Структура классов представляет иерархию наследования типа «обобщение-специализация», в которой подкласс (потомок) представляет собой специализированный частный случай своего суперкласса (предка). Пример: «токарь есть частный случай рабочего», «датчик температуры есть частный случай датчика».

Структура объектов представляет иерархию типа «агрегация», в которой один объект содержится внутри другого. Пример: «окно содержит полосы прокрутки, меню, строку состояния», «управляющий элемент в теплице содержит датчики температуры, влажности, кислотности и освещения».

Типизация – это способ защититься от использования объектов одного класса вместо другого, или, по крайней мере, управлять таким использованием. Типизация заставляет выражать абстракции так, чтобы язык программирования поддерживал соблюдение принятых проектных решений. Различают сильную и слабую типизации.

Параллелизм – это свойство, отличающее активные объекты от пассивных. Предполагает параллельную (независимую) работу ряда активных объектов. При проектировании надо принимать меры, гарантирующие, что объект не будет растерзан на части несколькими независимыми процессами.

Сохраняемость – способность объекта существовать во времени, переживая породивший его процесс, и (или) в пространстве, перемещаясь из своего первоначального адресного пространства.

Спектр сохраняемости объектов охватывает:

- промежуточные результаты вычисления выражений;
- локальные переменные в подпрограммах;
- собственные (глобальные) переменные и динамически создаваемые данные;
- данные, сохраняющиеся между сеансами выполнения программы;
- данные, сохраняемые при переходе на новую версию программы;
- данные, которые вообще переживают программу.

Традиционно, первыми тремя уровнями занимаются языки программирования, а последними – базы данных. Возможны смешанные решения: программисты разрабатывают специальные схемы для сохранения объектов в период между запусками программы, а конструкторы баз данных переиначивают свою технологию под короткоживущие объекты.

Существуют так называемые объектно-ориентированные базы данных (OODB), которые хранят не только данные, но и классы. Эта технология не прижилась, на практике предпочитают создавать объектно-ориентированную оболочку для реляционных баз данных.

Преимущества объектной модели

1. Позволяет в полной мере использовать возможности объектных и объектно-ориентированных языков программирования.
2. Повышает уровень унификации разработки и пригодность для повторного использования не только программ, но и проектов. Использование предыдущих разработок дает выигрыш в стоимости и времени.
3. Упрощает процесс внесения изменений из-за наличия стабильных промежуточных описаний. Это дает возможность не перерабатывать систему целиком даже в случае существенных изменений исходных требований.
4. Уменьшает вероятности допущения разнообразных ошибок.
5. Ориентирована на человеческое восприятие мира.

Некоторые факты из истории

1969 – Кэй (Kay) в своей диссертации предложил идею объектного подхода;

1979 – Джонс (Jones) ввел концепцию объектного подхода;

1981 – Буч (Booch) предложил методы OOD;

1988 – Шлеер и Меллор (Shlaer and Mellor) предложили методы ООА;

1988 – Страуструп (Stroustrup) опубликовал методическую статью по ООР;

1991 – Румбах (Rumbaugh) объединил ООА и OOD.

Организации, отвечающие за стандарты по объектной технологии: Object Management Group (OMG) и комитет ANSI X3J7.

1.2 Классы и объекты

Одними их базовых понятий объектной модели являются понятия классов и объектов.

Объект моделирует часть окружающей действительности и, таким образом, существует во времени и пространстве. Объект обладает состоянием, поведением и идентичностью; структура и поведение схожих объектов определяет общий для них класс; термины «экземпляр класса» и «объект» взаимозаменяемы.

Состояние объекта характеризуется перечнем (обычно статическим) всех свойств данного объекта и текущими (обычно динамическими) значениями каждого из этих свойств. Пример: автомат по продаже напитков.

Поведение определяет совокупность действий и реакций объекта; выражается в терминах состояния объекта и передачи сообщений. Иными словами, поведение объекта – это наблюдаемая и проверяемая извне деятельность. Состояние объекта представляет суммарный результат его поведения.

Операцией называется определенное воздействие одного объекта на другой с целью вызвать соответствующую реакцию; это услуга, которую класс может предоставить своим клиентам. Типичный клиент может совершать операции 5 видов:

1. *Модификатор* – операция изменения состояния объекта.
2. *Селектор* – операция, считывающая состояние объекта, но не изменяющая его.
3. *Итератор* – операция, позволяющая организовать доступ ко всем частям объекта в строго определенной последовательности.
4. *Конструктор* – операция создания объекта и/или его инициализации.
5. *Деструктор* – операция очищения и/или разрушения объекта.

Совокупность всех методов и свободных процедур, относящихся к конкретному объекту, образует *протокол* этого объекта.

Объекты могут быть активными и пассивными. Активный объект может проявлять свое поведение без воздействия со стороны других объектов. Пассивный объект, напротив, может изменять свое состояние только под воздействием других объектов. Таким образом, активные объекты системы – источники управляющих воздействий.

Идентичность – это такое свойство объекта, которое отличает его от всех других объектов. Следует уметь отличать имя объекта от самого объекта. Если абстракция представляет собой собрание свойств без собственного поведения, это не объект, а структура.

Система реализуется только в процессе взаимодействия объектов. Отношения между объектами могут быть двух типов: связи и агрегация.

Связь – это специфическое сопоставление, через которое клиент запрашивает услугу у объекта-сервера или через которое один объект находит путь к другому.

Участвуя в связи, объект может выполнять одну из трех ролей:

- **актер**: может воздействовать на другие объекты, но сам никогда не подвергается воздействию других объектов (исключительно активный объект);
- **сервер**: может только подвергаться воздействию со стороны других объектов, но сам никогда не выступает в роли воздействующего объекта (исключительно пассивный объект);
- **агент**: может выступать как в активной, так и в пассивной роли.

На рис.1 представлены четыре объекта, каждый из которых характеризует описанные выше роли.

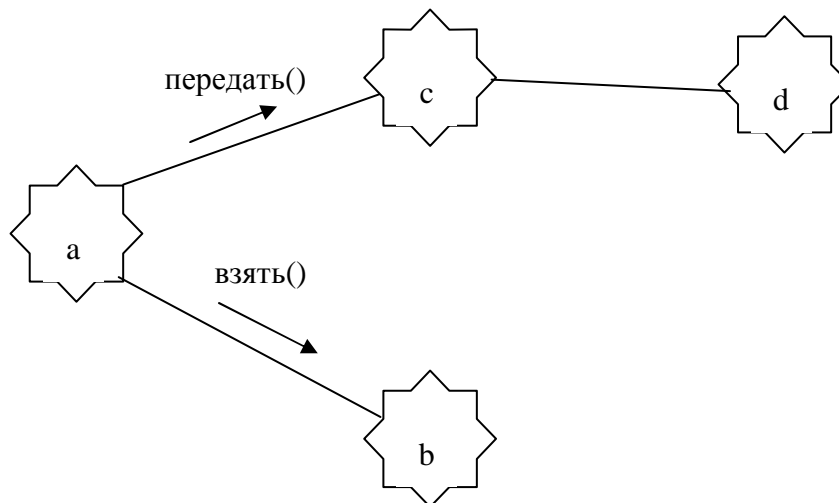


Рисунок 1 – Взаимодействующие объекты

Агрегация – специализированный частный случай ассоциации; описывает отношения целого и части, приводящие к соответствующей иерархии объектов.

Агрегация может означать физическое вхождение одного объекта в другой (автомобиль и двигатель), но не обязательно (акционер и акции). Пример агрегации с физическим вхождением представлен на рис. 2.



Рисунок 2 – Агрегация

Класс – это некое множество объектов, имеющих общую структуру и общее поведение. В то время как объект обозначает конкретную сущность, класс определяет лишь абстракцию существенного в объекте. Любой конкретный объект является просто экземпляром класса.

Поддерживаются 6 (шесть) видов отношений между классами.

1. **Ассоциация** – смысловая связь (по умолчанию – двусторонняя), фиксирующая участников, их роли и мощность отношения (1:1, 1:*, *:*)).

Товар ————— Сделка

2. **Наследование** – такое отношение между классами, когда один класс повторяет структуру и поведение другого или других классов. Устанавливает отношение общего и частного. Классы, экземпляры которых создаются, называются конкретными, не создаются – абстрактными. Выводы: у любого класса может быть два вида клиентов – экземпляры (объекты) и подклассы (наследники).

Изображается в виде стрелки, направленной от потомка к предку:

Датчик ←———— Датчик температуры

3. **Агрегация** между классами имеет тот же смысл, что и в случае объектов. Различают агрегацию *по значению* (физическое включение) и *по ссылке*. Если есть сомнения в выборе между наследованием и агрегацией, лучше выбрать вторую.

Изображается в виде линии с закрашенным кругом:

Контроллер ●———— Нагреватель

4. Отношение **использования** между классами соответствует связи между их экземплярами (клиент-серверные отношения).

Изображается в виде линии с пустым кругом:

Контроллер ○———— План выращивания

5. **Инстанцирование** предполагает использование параметризованных классов, когда в качестве формального параметра классу передается указатель на какой-либо класс, а при обращении к его экземпляру формальный параметр замещается фактическим. Является продолжением (или частным случаем) отношения использования.

Изображается в виде прямоугольника-врезки в правом верхнем углу класса.

6. **Метакласс** – это класс, экземпляры которого являются классами.

Изображается в виде закрашенного класса.

На этапе анализа и ранних стадиях проектирования решаются две основные задачи:

- выявление основных классов и объектов, составляющих словарь предметной области;
- построение структур, обеспечивающих взаимодействие объектов, при котором выполняются требования задачи.

Для оценки качества классов и объектов, выделяемых в системе, можно предложить следующие пять критериев: зацепление, связность, достаточность, полнота и примитивность.

Зацепление определяет степень глубины связей между отдельными модулями (чем меньше, тем лучше). Пример неправильного подхода: источник питания стереосистемы размещен в одной из колонок.

Связность – это степень взаимодействия между элементами отдельного модуля (класса, объекта), характеристика его насыщенности: «Класс Dog будет функционально связным, если он описывает поведение собаки, всей собаки, и ничего, кроме собаки».

Достаточность – наличие в классе или модуле всего необходимого для реализации логичного и эффективного поведения.

Под **полнотой** подразумевается наличие в интерфейсной части класса всех характеристик абстракции; интерфейс должен гарантировать все для взаимодействия с пользователями.

Примитивными являются только такие операции, которые требуют доступа к внутренней реализации абстракции

1.3 Классификация

Классификация – это средство упорядочения знаний. Исторически известны только **три** подхода: классическая категоризация, концептуальная кластеризация, теория прототипов.

При **классическом подходе** все вещи, обладающие данным свойством или совокупностью свойств, формируют некоторую категорию. Причем наличие этих свойств является необходимым и достаточным условием, определяющим категорию.

Однако естественные категории нечетко отделены друг от друга. Большинство птиц летает, но не все. Стул может быть деревянным, металлическим или пластмассовым, а число ножек необязательно равно четырем.

Концептуальная кластеризация – современный вариант классического подхода. Здесь сначала формируются концептуальные описания классов («кластеров объектов»), а затем мы классифицируем сущности в соответствии с их описаниями. Это именно понятие, а не признак или свойство: «Если песня скорее про любовь, чем про что-то другое, то мы помещаем ее в категорию *любовная песня*, хотя степень любовности едва ли можно измерить».

Концептуальную кластеризацию можно связать с теорией нечетких (многозначных) множеств, в которой объект может принадлежать к нескольким категориям одновременно с разной степенью точности.

Однако существуют некоторые абстракции, которые не имеют ни четких свойств, ни четкого определения. **Теория прототипов** определяет класс одним объектом-прототипом; новый объект относится к классу при условии, что он наделен существенным сходством с прототипом. Пример: игры.

На практике мы сначала идентифицируем классы и объекты по свойствам, важным в данной ситуации, то есть стараемся выделить и отобрать структуры и типы поведения с помощью словаря предметной области. Если таким путем не удалось построить нормальной структуры классов, мы пробуем концептуальный подход: уделяем внимание поведению объектов. Наконец, пробуем выделить прототипы и ассоциировать с ними объекты. Эти три способа классификации составляют теоретическую основу объектно-ориентированного анализа.

Границы между стадиями анализа и проектирования размыты, однако в процессе анализа мы моделируем проблему, *обнаруживая* классы и объекты, которые составляют словарь предметной области, в то время как на стадии проектирования мы *изобретаем* абстракции и механизмы, обеспечивающие требуемое поведение.

Существует **семь** проверенных практикой подходов к анализу объектно-ориентированных систем.

1. **Классические подходы** – опираются на классическую категоризацию.

Шлаер и Меллор: осязаемые предметы (датчики), роли (рабочий, студент), события (запрос, прерывание), взаимодействие (встреча, пересечение).

Росс (БД): люди (выполняют определенные функции), места (области, связанные с людьми или предметами), предметы (осязаемый материальный объект), организации (формально организованная совокупность людей и предметов, которая имеет определенную цель и не зависит от отдельных индивидуумов), концепции (принципы и идеи), события.

Коад и Йордан: структуры (отношения «целое-часть» и «общее-частное»), другие системы (внешние), устройства, события, роли, места, организационные единицы.

2. При **анализе поведения** именно динамическое поведение рассматривается как первоисточник классов и объектов; основан на концептуальной кластеризации. Инициаторы определенного поведения и его участники опознаются как объекты и делаются ответственными за определенные роли. Примеры: ввод/вывод, запрос.

3. **Анализ предметной области** – попытка выделить те объекты, операции и связи, которые эксперты данной области считают наиболее важными.

Этапы: построение скелетной модели предметной области; изучение существующих в данной области систем; определение сходства и различий между системами; уточнение общей модели для приспособления к нуждам конкретной системы.

Пример: предметная область – бухгалтерские отчеты; определяют абстракции и механизмы, обслуживающие все виды отчетов; цель исходной задачи – создание системы отчетов.

Эксперт – будущий пользователь системы (бухгалтер, диспетчер).

4. По отдельности все вышеперечисленные подходы сильно зависят от индивидуальных способностей и опыта аналитика. **Анализ вариантов** предусматривает упорядоченное последовательное их использо-

вание. Основан на перечислении и тщательной проработке сценариев работы системы.

5. **Метод CRC-карточек** удачно реализует на практике предыдущий подход (Class / Responsibilities / Collaborators – Класс / Ответственности / Участники). Сверху на карточке пишется название класса, снизу в левой половине – за что он отвечает, в правой – с кем он сотрудничает. Проход по сценарию приводит к дописыванию новых пунктов в существующих карточках или к созданию новых. Расположение карточек может показать поток сообщений между объектами и иерархию классов.

6. **Неформальное описание** – радикальная альтернатива классическому анализу (Аббот). Надо описать задачу или ее часть на обычном (человеческом) языке, а потом подчеркнуть существительные и глаголы. Существительные – кандидаты на роль классов, а глаголы могут стать именами операций. Метод можно автоматизировать. Используется в Токийском технологическом институте и в Fujitsu.

7. Вторая альтернатива классической технике – использование **структурного анализа** как основы объектно-ориентированного проектирования. После его проведения мы уже имеем модель системы, описанную диаграммами потоков данных; исходя из этого, можно приступить к определению классов и объектов любыми другими способами.

2 УНИФИЦИРОВАННЫЙ ЯЗЫК МОДЕЛИРОВАНИЯ UML КАК СРЕДСТВО ПРОЕКТИРОВАНИЯ ПРОГРАММНЫХ СИСТЕМ И БИЗНЕС-ПРОЦЕССОВ

Унифицированный язык моделирования UML (Unified Modeling Language) предназначен для описания, визуализации и документирования объектно-ориентированных программных систем и бизнес-процессов с ориентацией на их последующую реализацию в виде программного обеспечения.

Используется в следующих областях:

- информационные системы предприятий;
- банковская и финансовая деятельность;
- телекоммуникации;

- транспорт;
- авиация и космонавтика;
- торговля;
- распределенные Web-системы.

Мы будем рассматривать UML как средство проектирования и документирования при разработке информационных систем (как в виде новых программных продуктов, так и в виде компьютерного обеспечения бизнес-процессов) для промышленных и торговых предприятий, а также банковских и образовательных учреждений.

2.1 Предыстория, этапы развития и общая структура UML

Можно выделить методологические и математические основы UML. Кроме того, UML базируется на диаграммах структурного системного анализа.

Методологические основы UML

- методология процедурно-ориентированного программирования;
- методология объектно-ориентированного программирования;
- методология объектно-ориентированного анализа и проектирования;
- методология системного анализа и системного моделирования

Математические основы UML

- теория множеств (диаграммы Венна);
- теория графов;
- семантические сети

Диаграммы структурного системного анализа

- диаграммы «сущность-связь» (ERD – Entity-Relationship Diagrams);
- диаграммы функционального моделирования (SADT – Structured Analysis and Design Technique);
- диаграммы потоков данных (DFD – Data Flow Diagrams)

Отдельные языки объектно-ориентированного моделирования стали появляться к концу 1970-х годов; к 1994-му году их число достигло

50. Каждый разработчик считал свой метод лучшим; принятие отдельных методик (IDEF0) в качестве стандартов не смогло изменить сложившуюся ситуацию.

В 1994-м году самыми распространенными методами становятся:

- метод Гради Буча (Grady Booch) – Booch'93;
- метод Джеймса Румбаха (James Rumbaugh) – Object Modeling Technique (OMT-2);
- метод Айвара Джекобсона (Ivar Jacobson) – Object-Oriented Software Engineering (OOSE).

В октябре 1994 года Г.Буч и Дж.Румбах из Rational Software начали работу по *унификации* своих методов, позже к ним присоединился А.Джекобсон из Objectory AB (Швеция). Работа была направлена на получение нового метода, который бы содержал все лучшее из предыдущих, а именно:

- позволял моделировать не только программное обеспечение, но и более широкие классы систем и бизнес-приложений, с использованием объектно-ориентированных понятий;
- явным образом обеспечивал взаимосвязь между базовыми понятиями для моделей концептуального и физического уровней;
- обеспечивал масштабируемость моделей, что крайне необходимо для сложных многоцелевых систем;
- был понятен аналитикам и программистам, а также поддерживался специальными инструментальными средствами, реализованными на различных компьютерных платформах.

Полученный результат был назван UML-0.8 и опубликован в октябре 1995 года, после чего был передан практически всем компаниям – разработчикам программного обеспечения для замечаний и дополнений. Версия UML-1.0 появилась на свет в начале 1997 года; в марте 2003 года была выпущена версия UML-1.5.

Хотя в настоящее время в странах СНГ используются три нотации визуального моделирования: IDEF (Icam DEFinition), ARIS (Architecture of Integrated Information Systems) и UML, именно последняя постепенно становится стандартом при разработке информационных систем. Ряд сред разработки приложений и визуального программирования – MS

Visual Studio.NET, Borland Delphi 2005 и т.п. – не только поддерживают нотацию UML в качестве средства моделирования, но и позволяют получить исполнимые программы на основе разработанной модели.

Основные компоненты UML

Язык UML опирается на некоторый набор базовых принципов, применяемых при моделировании сложных систем, а именно:

- *принцип абстрагирования*, предписывающий включать в модель только те аспекты проектируемой системы, которые имеют непосредственное отношение к выполнению системой своих функций;
- *принцип многомодельности*, утверждающий, что никакая единственная модель не может с достаточной степенью адекватности описывать различные аспекты сложной системы;
- *принцип иерархического построения*, предписывающий рассматривать процесс построения модели на разных уровнях абстрагирования или детализации в рамках фиксированных представлений.

Таким образом, процесс ООАП можно представить как поуровневый спуск от наиболее общих моделей и представлений концептуального уровня к более частным и детальным представлениям логического и физического уровней.

Общая схема взаимосвязей моделей и представлений сложной системы в процессе объектно-ориентированного анализа и проектирования представлена на рис. 3.



Рисунок 3 – Схема взаимосвязей моделей и представлений сложной системы

Формальное описание языка UML основывается на некоторой общей иерархической структуре модельных представлений, состоящей из четырех уровней:

- мета-мета-модель;
- мета-модель;
- модель;
- объекты пользователя.

Подробнее о каждом уровне см. в [1-7].

В рамках языка UML все представления о модели системы фиксируются в виде специальных графических конструкций, получивших название диаграмм. Предусмотрены следующие виды диаграмм (рис.4):

- 1 Диаграмма вариантов использования (use case diagram)
- 2 Диаграмма классов (class diagram)
- 3 Диаграммы поведения (behavior diagrams):
 - 3.1 Диаграмма состояний (statechart diagram);
 - 3.2 Диаграмма деятельности (activity diagram)
 - 3.3 Диаграммы взаимодействия (interaction diagrams):
 - 3.3.1 Диаграмма последовательности (sequence diagram);

3.3.2 Диаграмма кооперации (collaboration diagram)

4 Диаграммы реализации (implementation diagrams):

4.1 Диаграмма компонентов (component diagram);

4.2 Диаграмма развертывания (deployment diagram).



Рисунок 4 – Модель сложной системы в нотации UML

Каждая из диаграмм по-своему описывает (конкретизирует) систему, причем общая модель может содержать лишь те диаграммы, которые достаточно адекватно характеризуют проектируемую систему.

Особенности изображения диаграмм

Большинство перечисленных диаграмм являются, по сути, графами специального вида, состоящими из вершин в форме геометрических фигур, которые связаны между собой ребрами или дугами. Как правило, ни размеры, ни расположение (за исключением диаграммы последовательностей) элементов диаграмм не имеют принципиального значения.

Для всех диаграмм существует три типа визуальных графических обозначений:

- геометрические фигуры на плоскости, играющие роль вершин графов (*графические примитивы*);
- графические взаимосвязи, представляемые различными линиями;
- специальные символы (графические или текстовые), изображаемые вблизи других элементов и называемые *спецификациями*.

При изображении диаграмм следует придерживаться следующих семи рекомендаций:

1. Каждая диаграмма должна служить законченным представлением соответствующего фрагмента моделируемой предметной области.
2. Все сущности на каждой диаграмме должны быть одного концептуального уровня (для сложных моделей следует использовать принципы последовательного уточнения или детализации).
3. Вся информация о сущностях должна быть явно представлена на диаграмме (не следует во всем полагаться на «значения по умолчанию»).
4. Диаграммы не должны содержать противоречивой информации.
5. Диаграммы не должны быть перегружены текстовой информацией.
6. Количество типов диаграмм для конкретной модели не является строго фиксированным: например, модель может не содержать диаграмму развертывания, если предполагается исключительно локальная работа системы.
7. Модель системы должна содержать только те элементы, которые определены в нотации языка UML.

2.2 Диаграмма концептуального моделирования – диаграмма вариантов использования (use case diagram)

Диаграмма вариантов использования (иногда ее называют диаграммой прецедентов) описывает функциональное назначение системы или, другими словами, то, что система будет делать в процессе своего функционирования. Эта диаграмма является исходным концептуальным

представлением или концептуальной моделью системы в процессе ее проектирования и разработки.

Разработка диаграммы вариантов использования преследует такие цели:

- определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы;
- сформулировать общие требования к функциональному поведению проектируемой системы;
- разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей;
- подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в форме так называемых вариантов использования, с которыми взаимодействуют некоторые внешние сущности, или актеры. При этом *актером*, или *действующим лицом*, называется любой объект, субъект или система, взаимодействующая с моделируемой системой извне. В свою очередь, вариант использования служит для описания сервисов, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером.

Таким образом, базовыми элементами диаграммы вариантов использования являются: собственно вариант использования (прецедент), актер и примечания.

Вариант использования (use case) представляет собой спецификацию общих особенностей поведения или функционирования моделируемой системы без рассмотрения внутренней структуры этой системы. Обозначается на диаграмме эллипсом, внутри (или ниже) которого содержится его краткое название или имя в форме глагола с пояснительными словами; при этом текст обязательно должен начинаться с заглавной буквы (рис.5).

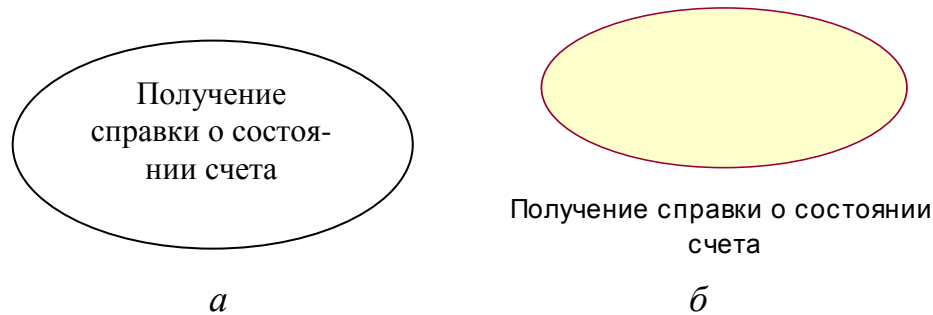


Рисунок 5 – Графическое обозначение варианта использования в исходной нотации (а) или в среде Rational Rose (б)

Актёр (actor) представляет собой любую внешнюю по отношению к моделируемой системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или решения частных задач. Стандартным графическим обозначением актера на диаграммах является фигурка «человечка», под которой записывается конкретное имя актера (рис. 6). Как правило, под именем подразумевается должность (если актер – человек), но никак не имя собственное (Вася Иванов) или название конкретного устройства (Маршрутизатор Cisco 3640).

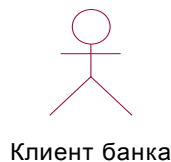


Рисунок 6 – Графическое обозначение актера

Актеры взаимодействуют с системой посредством передачи и приема сообщений от вариантов использования. Сообщение представляет собой запрос актером сервиса от системы и получение этого сервиса. Это взаимодействие может быть выражено посредством ассоциаций между отдельными актерами и вариантами использования или классами. Кроме этого, с актерами могут быть связаны интерфейсы, которые определяют, каким образом другие элементы модели взаимодействуют с этими актерами.

Интерфейс (interface) служит для спецификации параметров модели, которые видимы извне без указания их внутренней структуры. В языке UML интерфейс является классификатором и характеризует только ограниченную часть поведения моделируемой сущности. Применительно к диаграммам вариантов использования, интерфейсы определяют совокупность операций, которые обеспечивают необходимый набор сервисов или функциональности для актеров. Интерфейсы не могут содержать ни атрибутов, ни состояний, ни направленных ассоциаций. Они содержат только операции без указания особенностей их реализации. Формально интерфейс эквивалентен абстрактному классу без атрибутов и методов с наличием только абстрактных операций. На диаграмме вариантов использования интерфейс изображается в виде маленького круга, рядом с которым записывается его имя. В качестве имени может быть существительное, которое характеризует соответствующую информацию или сервис (рис. 7).



Интерфейс

Рисунок 7 – Графическое обозначение интерфейса

Примечание (note) предназначено для включения в модель произвольной текстовой информации, имеющей непосредственное отношение к контексту разрабатываемого проекта. Графически изображается прямоугольником с «загнутым» уголком (рис.8), соединенным с элементом диаграммы пунктирной линией. Если примечание имеет ключевое слово <<constraint>>, то является ограничением, накладываемым на элемент.

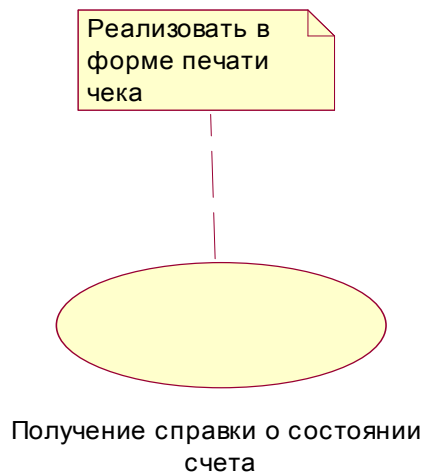


Рисунок 8 – Пример примечания

Между компонентами диаграммы вариантов использования могут существовать различные отношения, которые описывают взаимодействие экземпляров одних актеров и вариантов использования с экземплярами других актеров и вариантов.

В языке UML имеется несколько стандартных видов отношений:

- отношение ассоциации (association relationship);
- отношение включения (include relationship);
- отношение расширения (extend relationship);
- отношение обобщения (generalization relationship).

Отношение ассоциации определяет особенности взаимодействия актеров и вариантов использования и обозначается отрезком сплошной линии (рис. 9).

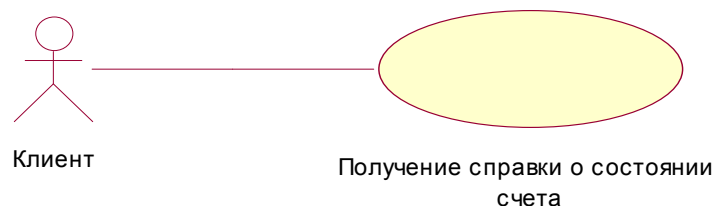


Рисунок 9 – Пример отношения ассоциации

Отношение включения между двумя вариантами использования указывает, что некоторое заданное поведение для одного варианта использования включается в качестве составного компонента в последовательность поведения другого варианта использования. Отношение включения, направленное от варианта использования А к варианту использо-

вания В, указывает, что каждый экземпляр варианта А включает в себя функциональные свойства, заданные для варианта В. Эти свойства специализируют поведение соответствующего варианта А на данной диаграмме. Графически данное отношение обозначается пунктирной линией со стрелкой (вариант отношения зависимости), направленной от базового варианта использования к включаемому, и помечается ключевым словом <<include>> (рис. 10). Один вариант использования может быть включен в несколько других вариантов, а также включать в себя другие варианты.

Отношение расширения определяет взаимосвязь базового варианта использования с некоторым другим вариантом использования, функциональное поведение которого задействуется не всегда, а только при выполнении некоторых дополнительных условий.

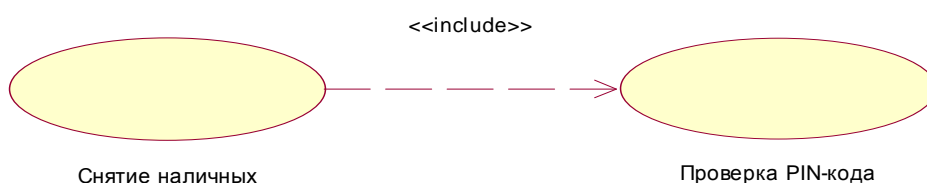


Рисунок 10 – Пример отношения включения

Графически обозначается пунктирной линией со стрелкой (вариант отношения зависимости), направленной от того варианта использования, который является расширением для исходного варианта использования, и помечается ключевым словом <<extend>> (рис. 11).

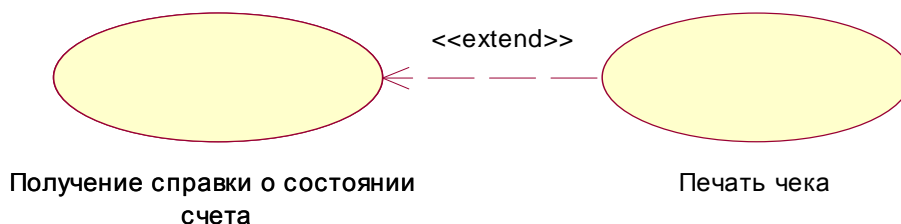


Рисунок 11 – Пример отношения расширения

Отношение обобщения служит для указания того факта, что некоторый элемент А может быть обобщен до элемента В. Таким образом, А будет являться специализацией В. При этом В называется предком или родителем по отношению А, а А – потомком по отношению к В. Графи-

чески данное отношение обозначается сплошной линией со стрелкой в форме не закрашенного треугольника, которая указывает на родительский элемент. Следует отметить, что данный вид отношений применяется как между вариантами использования (рис. 12), так и между актерами (рис. 13).

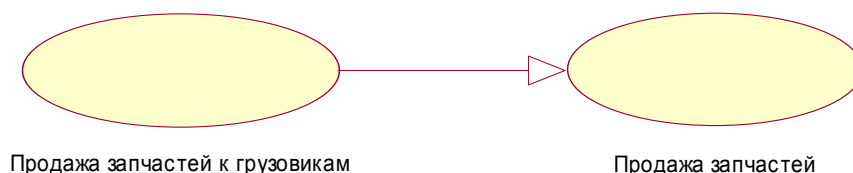


Рисунок 12 – Пример отношения обобщения между вариантами использования



Рисунок 13 – Пример отношения обобщения между актерами

В [2] представлен пример модели системы работы банкомата, которую мы также примем в качестве базового примера. Рассматриваемая система имеет двух актеров – Клиент и Банк, причем главным является Клиент, поскольку инициирует работу. Базовые варианты использования – «Снятие наличных с карточки» и «Получение информации о состоянии счета». Дополнительные сервисы – «Проверка PIN-кода» (используется всегда, поэтому связано отношением включения) и «Печать чека» (используется при желании Клиента увидеть состояние счета не только на экране, но и в печатном виде, поэтому связано отношением расширения). Полученная диаграмма представлена на рис. 14.

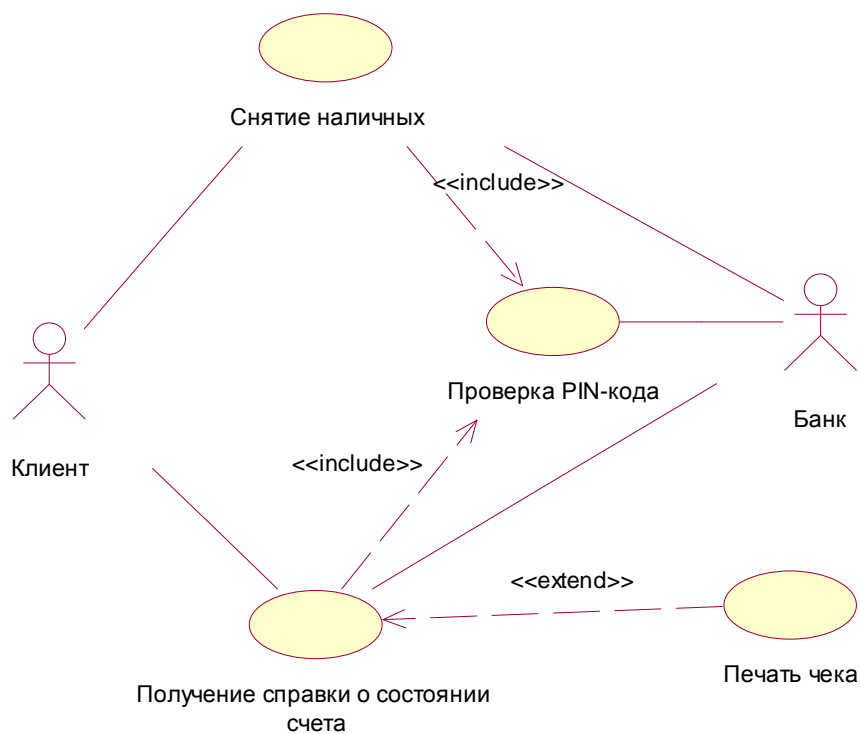


Рисунок 14 – Диаграмма вариантов использования для модели банкомата

Простейшая информационная система – программа для обработки какой-либо информации – может иметь двух актеров (обычного пользователя и администратора) и два базовых варианта использования («Ввод и модификация данных» и «Обработка данных»). Кроме того, возможны дополнительные сервисы – «Проверка имени и пароля» (используется постоянно для отличия простого пользователя от администратора, поэтому связано отношением включения) и «Формирование отчета» (используется пользователем при необходимости, поэтому связано отношением расширения). Полученная диаграмма представлена на рис. 15.

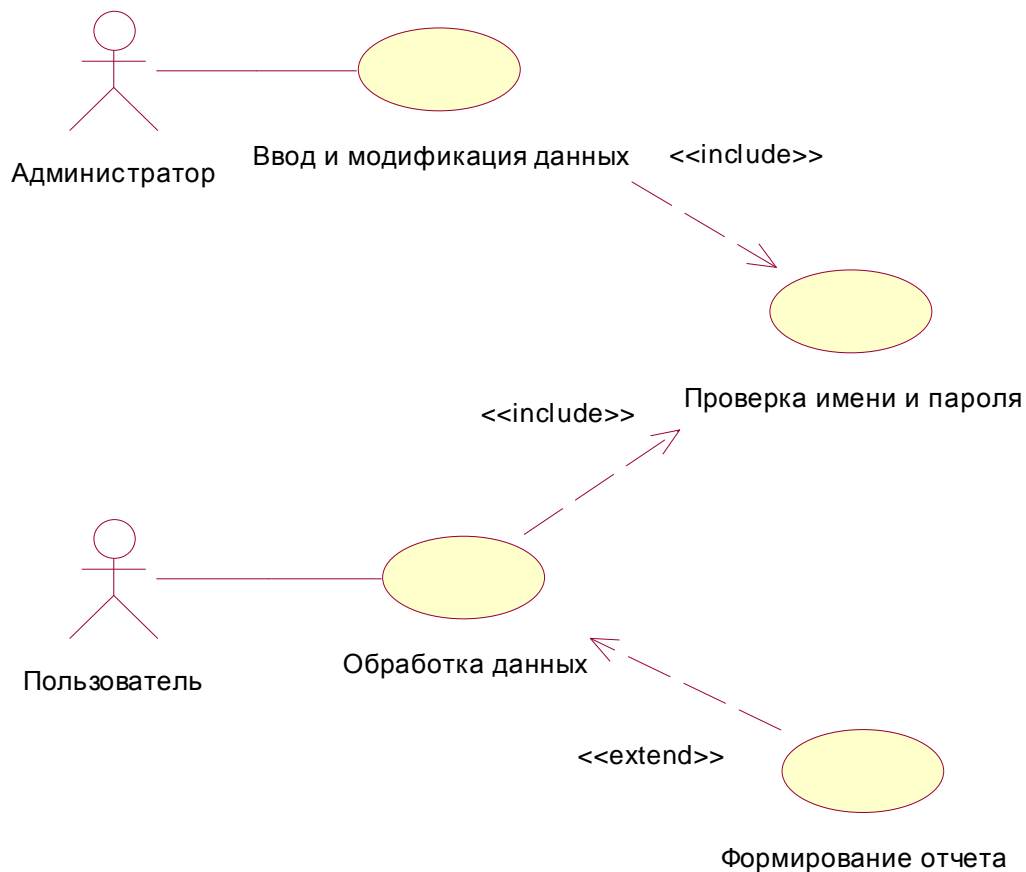


Рисунок 15 – Диаграмма вариантов использования для модели простейшей информационной системы

2.3 Диаграммы логического моделирования

Следующие пять диаграмм – классов, кооперации, последовательности, состояний и деятельности – описывают логический уровень модели (как статические аспекты структуры, так и динамические аспекты функционирования). Основные элементы этих диаграмм – классы, объекты и отношения между ними.

2.3.1 Диаграмма классов (*class diagram*)

Центральное место в методологии ООАП занимает разработка логической модели системы в виде диаграммы классов.

Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать, в част-

ности, различные взаимосвязи между отдельными сущностями предметной области, такими, как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений. На данной диаграмме не указывается информация о временных аспектах функционирования системы.

Базовыми элементами диаграммы классов являются классы сами (со своими атрибутами и операциями) и отношения между ними.

Класс – абстрактное описание или представление свойств множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов – графически изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции, в которых могут указываться имя класса, атрибуты (переменные) и операции (методы) (рис. 16).

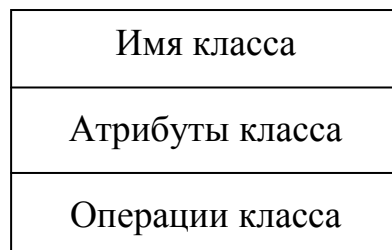


Рисунок 16 – Изображение класса

Как правило, если даже какая-то секция оказывается пустой, в обозначении класса она все равно показывается. Различные примеры изображения классов представлены на рис. 17



Рисунок 17 – Примеры изображений различных классов

Имя класса должно быть уникальным в пределах пакета (в пакете может содержаться несколько диаграмм). Записывается по центру секции полужирным шрифтом и должно начинаться с заглавной буквы. Если класс не может иметь экземпляров (объектов), то есть является абстрактным, его имя выделяется курсивом.

При обозначении имен класса рекомендуется использовать существительные, записанные без пробелов. Необходимо помнить, что имена классов образуют словарь предметной области при ООАП.

Атрибут класса служит для представления отдельного свойства или признака, который является общим для всех объектов данного класса. Запись атрибута подчиняется синтаксическим правилам:

<квантор видимости> <имя атрибута> [кратность]:
<тип атрибута> = <исходное значение> {строка-свойство}

Квантор видимости (visibility) может принимать одно из четырех значений:

- «+» – public – доступность без ограничений;
- «#» – protected – доступность только для данного класса и его потомков;
- «-» – private – доступность только для данного класса;
- «~» – package – доступность только в пределах данного пакета.

Имя атрибута должно начинаться со строчной (малой) буквы и не может содержать пробелов.

Кратность характеризует общее количество атрибутов данного типа, входящих в состав класса (по умолчанию равно 1), например:

- [0..1] – либо такого атрибута нет, либо он есть;
- [0..*] – либо такого атрибута нет, либо их сколько угодно;
- [1..5] – таких атрибутов может быть от 1 до 5.

Тип атрибута определяется типом данных, например: «цвет: Color» или «имяСотрудника [1..2]: String».

Исходное значение служит для задания некоторого начального значения атрибута в момент создания экземпляра класса, например:

«цвет:Color=(255,0,0)» или «имяСотрудника[1..2]:String=“Иван Иванович”».

Строка-свойство служит для указания дополнительных свойств атрибута, например: «заработнаяПлата:Деньги=500{frozen}» – фиксированная сумма.

Операция класса – это некоторый сервис, который предоставляет каждый экземпляр (объект) класса по требованию своих клиентов (других объектов, в том числе и экземпляров данного класса). Совокупность операций характеризует функциональный аспект поведения всех объектов данного класса.

Формат записи операции следующий:

<квантор видимости> <имя операции> (список параметров):
<возвращаемое значение> {строка-свойство}

Например:

+нарисовать (форма: Многоугольник = Прямоугольник)
–изменитьСчет (номерСчета: Integer): Деньги

Подробнее об операциях см. в [1-7].

Между классами могут быть следующие виды отношений:

- отношение ассоциации (association relationship);
- отношение обобщения (generalization relationship).
- отношение агрегации (aggregation relationship);
- отношение композиции (composition relationship);
- отношение зависимости (dependency relationship).

Отношение ассоциации соответствует наличию произвольной взаимосвязи между классами и может быть как ненаправленной (рис. 18), так и направленной (рис. 19). Если связаны два класса друг с другом, ассоциация называется *бинарной*, если класс связан сам с собой – *рефлекс-*

сивной. Ассоциация может иметь имя и содержать кратности классов-ролей ассоциации.

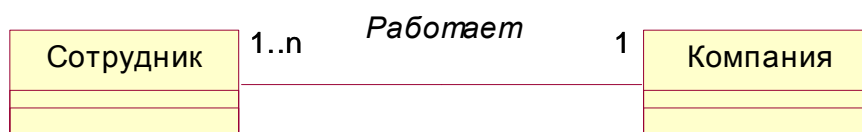


Рисунок 18 – Пример ненаправленной бинарной ассоциации



Рисунок 19 – Пример направленной бинарной ассоциации

Отношение обобщения представляет собой отношение между предком и потомком (рис. 20).

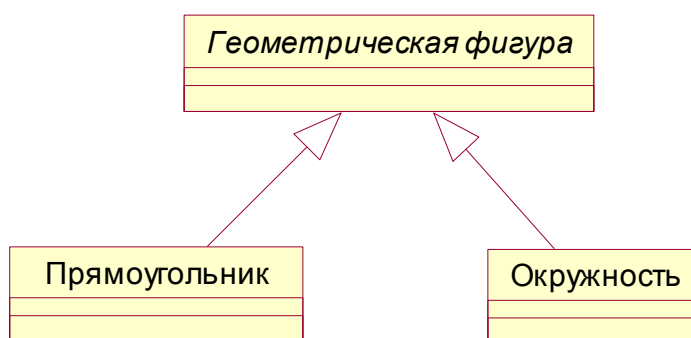


Рисунок 20 – Пример отношения обобщения

К стрелке-обобщению могут быть добавлены следующие ограничения:

{complete} – означает, что в данном отношении определены все классы-потомки, и других у данного класса-предка быть не может;

{incomplete} – означает, что, напротив, указаны не все классы-потомки данного класса-предка;

{disjoint} – означает, классы-потомки не могут содержать объектов, одновременно являющихся экземплярами двух и более классов;

{overlapping} – означает, что, напротив, отдельные экземпляры классов-потомков могут принадлежать одновременно нескольким классам.

Отношение агрегации имеет место между классами в том случае, если один из классов представляет собой некоторую сущность, которая включает в себя в качестве составных частей другие сущности (иерархия вида «часть – целое»). Изображается в виде отрезка линии с не закрашенным ромбом со стороны класса-контейнера (рис. 21).

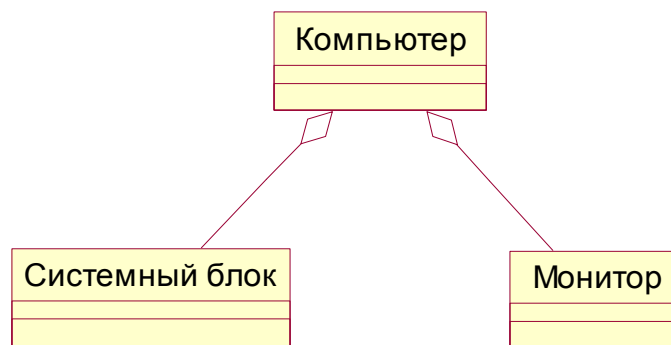


Рисунок 21 – Пример отношения агрегации

Отношение композиции представляет собой частный случай агрегации, при котором составные части не могут существовать в отрыве от целого. Изображается в виде отрезка линии с закрашенным ромбом со стороны класса-контейнера (рис. 22).

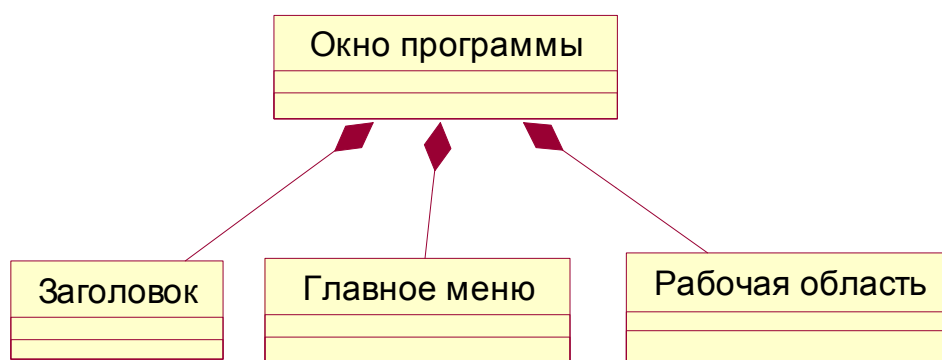


Рисунок 22 – Пример отношения композиции

Отношение зависимости используется в случае наличия особых отношений между элементами, которые нельзя точно отнести ни к одно-

му из вышеперечисленных. Изображается в виде пунктирной линии со стрелкой.

Интерфейс является особым случаем класса, у которого имеются только операции и отсутствуют атрибуты. Для изображения интерфейса используется стереотип <<interface>> или специальный графический символ – окружность.

На рис. 23 представлена диаграмма классов системы управления банкоматом из [2]. У ряда классов указаны их стереотипы: «Устройство чтения», «Экран банкомата», «Принтер банкомата» и «Устройство выдачи наличных» – <<boundary>>, «Контроллер банка» – <<interface>>, «Контроллер банкомата» – <<control>>.

Довольно часто классы на диаграмме содержат очень много атрибутов и операций, в результате чего ее просмотр становится затруднительным. В таких случаях ее делят на две части – собственно диаграмму классов (или диаграмму ассоциаций классов) с опущенными секциями атрибутов и операций (рис. 24) и диаграмму конкретизации классов (рис. 25).



Рисунок 23 – Диаграмма классов

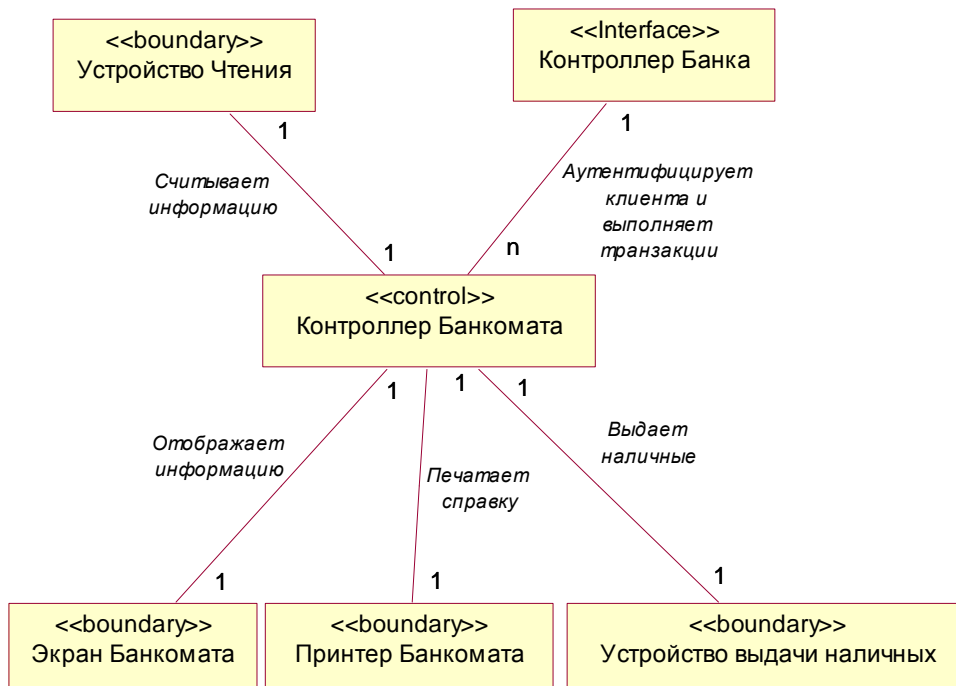


Рисунок 24 – Упрощенная диаграмма классов

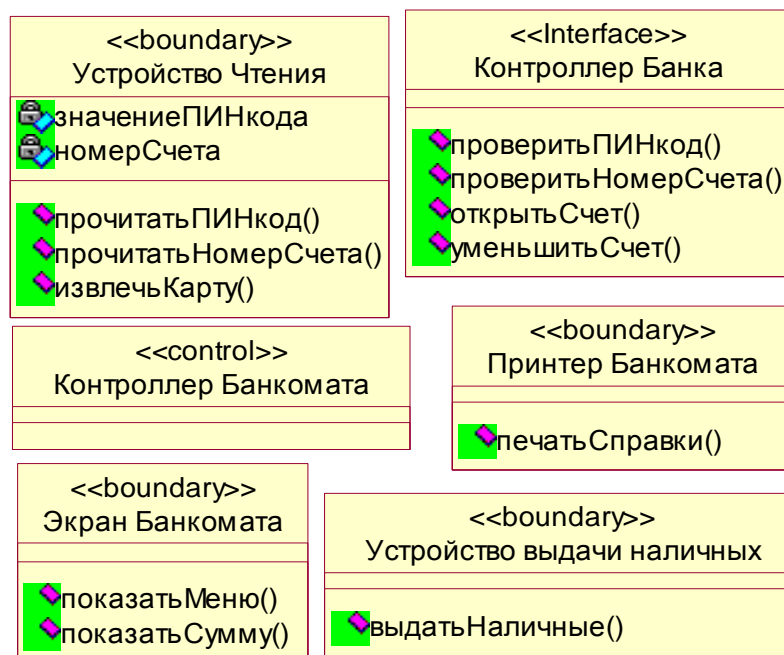


Рисунок 25 – Диаграмма конкретизации классов

На рис. 26 представлена диаграмма классов простейшей информационной системы. Здесь имеется два класса актеров – «Пользователь» и «Администратор», управляющий (<<control>>) класс «Программа» и два класса для работы с данными – «База данных» и «Отчет».

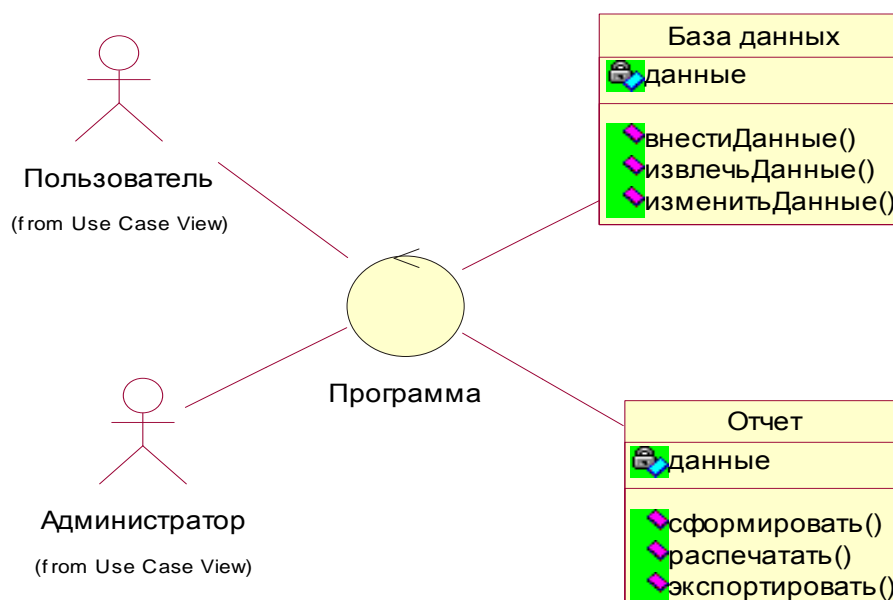


Рисунок 26 – Диаграмма классов для модели простейшей информационной системы

2.3.2 Диаграмма кооперации (*collaboration diagram*)

Для моделирования взаимодействия объектов в языке UML используются так называемые диаграммы взаимодействия, к которым относятся диаграмма кооперации и диаграмма последовательности.

Особенности передачи и приема сообщений в контексте статической структуры модели отображает диаграмма кооперации. Здесь поведение системы описывается на уровне отдельных объектов, которые обмениваются между собой сообщениями, чтобы достичь нужной цели или реализовать некоторый вариант использования. С точки зрения аналитика или архитектора системы важно представить в проекте структурные связи отдельных объектов между собой. Такое представление структуры модели как совокупности взаимодействующих объектов и обеспечивает данная диаграмма. Следует понимать, что одна и та же совокупность объектов может участвовать в реализации различных коопераций – при этом, в зависимости от рассматриваемой кооперации, могут изменяться как связи между отдельными объектами, так и поток сообщений между ними.

Ключевые понятия диаграммы кооперации – объекты (экземпляры классов), связи (экземпляры ассоциаций) и сообщения. Кооперация может быть представлена на двух уровнях:

- на уровне спецификации – показывает роли классификаторов и роли ассоциаций в рассматриваемом взаимодействии;
- на уровне примеров (экземпляров) – указывает объекты и связи, образующие отдельные роли в кооперации.

При моделировании информационных систем чаще всего используется второй случай, поэтому в настоящем пособии он и будет рассмотрен. Интересующиеся диаграммами уровня спецификации могут обратиться к [2].

Объект (*object*) является отдельным экземпляром класса, который создается на этапе реализации модели (выполнения программы). Он может иметь собственное имя и конкретные значения атрибутов. Соответственно, изображается в виде прямоугольника, иногда (при необходимости указания значений атрибутов) состоящего из двух секций (см. рис. 27).

Имя объекта
Значения атрибутов объекта

Рисунок 27 – Изображение объекта

Имя объекта в общем случае имеет следующий формат:

<собственное имя>/<имя роли классификатора>:<имя классификатора>

В отдельных случаях собственное имя объекта может отсутствовать – такой объект называется *анонимным* (двоеточие перед именем класса должно остаться обязательно). Если отсутствует имя класса, то объект называется *сиротой*.

Различные примеры изображения объектов представлены на рис.28.

<u>квадрат</u>	<u>квадрат : Прямоугольник</u>	<u>:Прямоугольник</u>
<u>квадрат : Прямоугольник</u> вершина=(10,10) сторона=5 цветЛинии=черный цветЗаливки=белый		<u>клиент / Инициатор запроса</u> <u>a1 / Обработчик : Сервер</u>

Рисунок 28 – Примеры изображений различных объектов

Иногда необходимо отдельно изобразить на диаграмме множество объектов, которые могут быть образованы на основе одного класса. Такое изображение в виде «наложенных» прямоугольников называется *мультиобъектом*. При этом может быть явно указано отношение агрегации (композиции) между мультиобъектом и отдельным объектом из его множества (рис. 29).

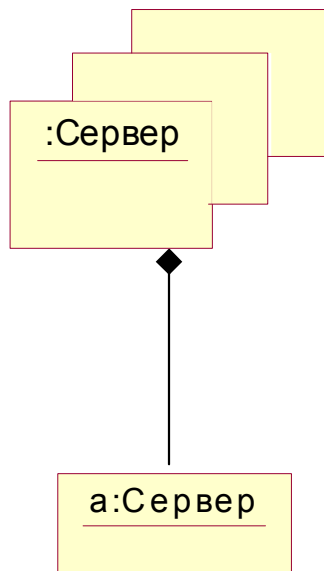


Рисунок 29 – Пример изображения мультиобъекта

Все объекты делятся на две категории: пассивные и активные. *Пассивный объект* (passive object) оперирует только данными и не может инициировать деятельность по управлению другими объектами. *Активный объект* (active object) имеет свой собственный поток управления (процесс) и может инициировать деятельность по управлению другими объектами. Как правило, активные объекты на диаграмме кооперации обозначаются либо утолщением границ прямоугольника, либо явным указанием ключевого слова {active}.

В качестве иллюстрации вышеизложенного на рис. 30 приведен пример диаграммы кооперации для вызова функции печати из текстового редактора [2].

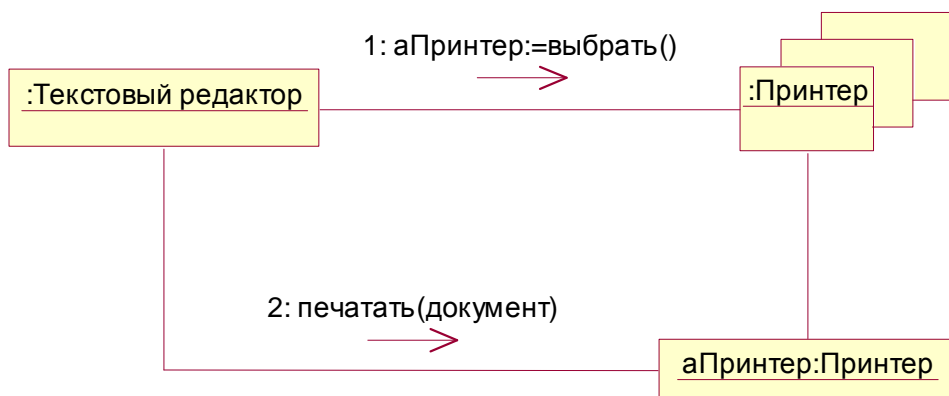


Рисунок 30 – Пример диаграммы кооперации

Составной объект (composite object) представляет экземпляр класса-композиции, который связан отношением композиции со своими частями (см. пример на рис. 31).

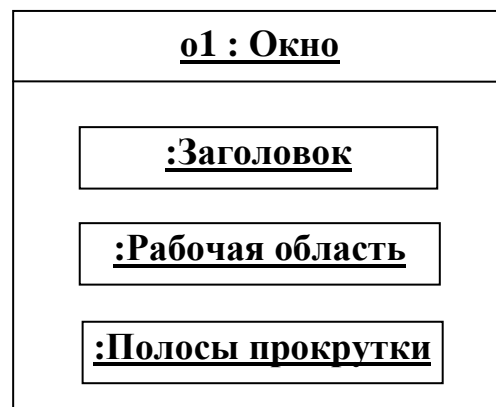


Рисунок 31 – Составной объект «Окно»

Связь (link) является экземпляром ассоциации и может иметь место между двумя и более объектами. Изображается отрезком прямой линии, на концах которого могут быть явно указаны имена ролей соответствующей ассоциации.

Сообщение (message) показывает коммуникацию между двумя объектами, один из которых передает другому некоторую информацию, при этом предполагается, что после получения сообщения вторым объектом последует некоторое действие. Сообщения изображаются стрелками рядом с соответствующей связью, направление стрелки указывает на получателя сообщения (см. рис. 30).

Стрелки сообщений могут быть трех типов:

- сплошная линия с треугольной закрашенной стрелкой (—►) обозначает вызов процедуры (операции) или передачу потока управления, при этом объект, передающий сообщение, ожидает окончания некоторых действий, вызванных этим сообщением (*синхронные сообщения*);
- сплошная линия с V-образной стрелкой (—>) обозначает *асинхронное сообщение*, то есть объект, передающий сообщение, продолжает свою деятельность, не ожидая ответа;

– пунктирная линия с V-образной стрелкой (----->) обозначает возврат из вызова процедуры; как правило, предполагаются по умолчанию и на диаграммах изображаются редко.

Сообщение имеет следующий формат:

<Предшествующие сообщения> <Выражение> <Значение:=имя>
<(Список аргументов)>

Смысл указания *предшествующих сообщений* заключается в том, что данное сообщение не может быть передано, пока не будут переданы адресатам перечисленные сообщения:

2,4 / 5: передать (данные)

Выражение может представлять собой условие, при котором будет осуществляться передача сообщения:

6 [(xc > 0) & (xc < 800)]: нарисоватьКруг (xc, 100)

Значение предполагает, что ожидается возврат некоторых результатов выполнения операции:

7: aПринтер:=выбратьПринтер()

На рис. 32 и рис. 33 приведены примеры диаграмм кооперации – для модели системы управления банкоматом (из [2] – прецедент «Снятие наличных») и для модели простейшей информационной системы (прецедент «Работа пользователя: формирование отчета»). Стереотипы связей (<<Local>>, <<Global>> и т. п.) показаны буквами на концах.

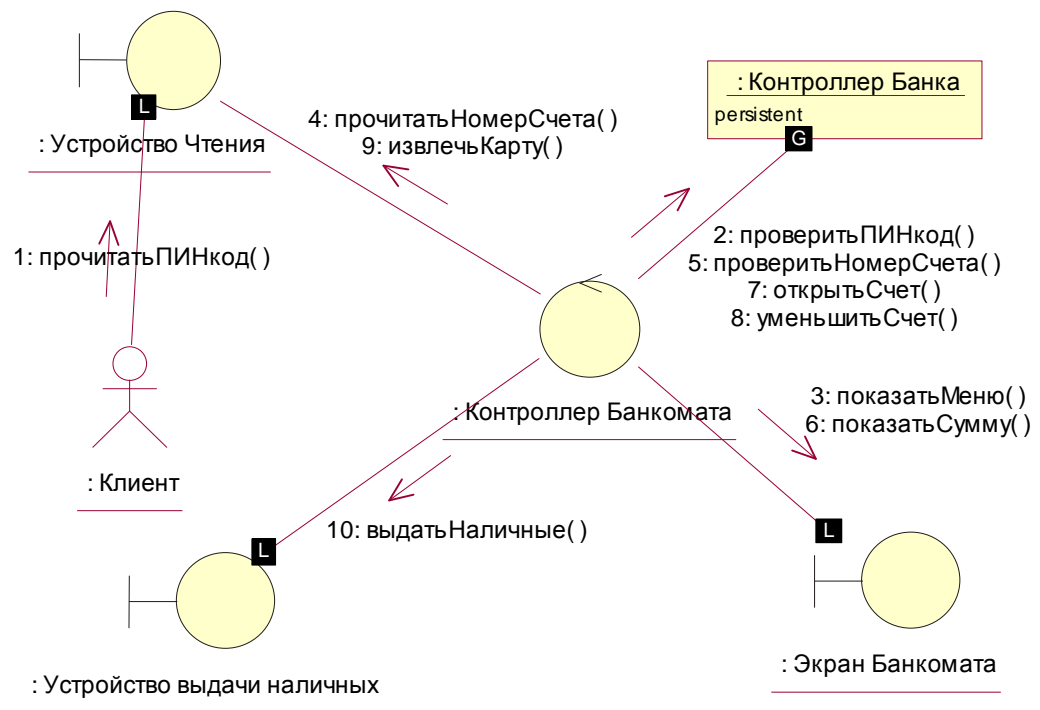


Рисунок 32 – Диаграмма кооперации для модели системы управления банкоматом

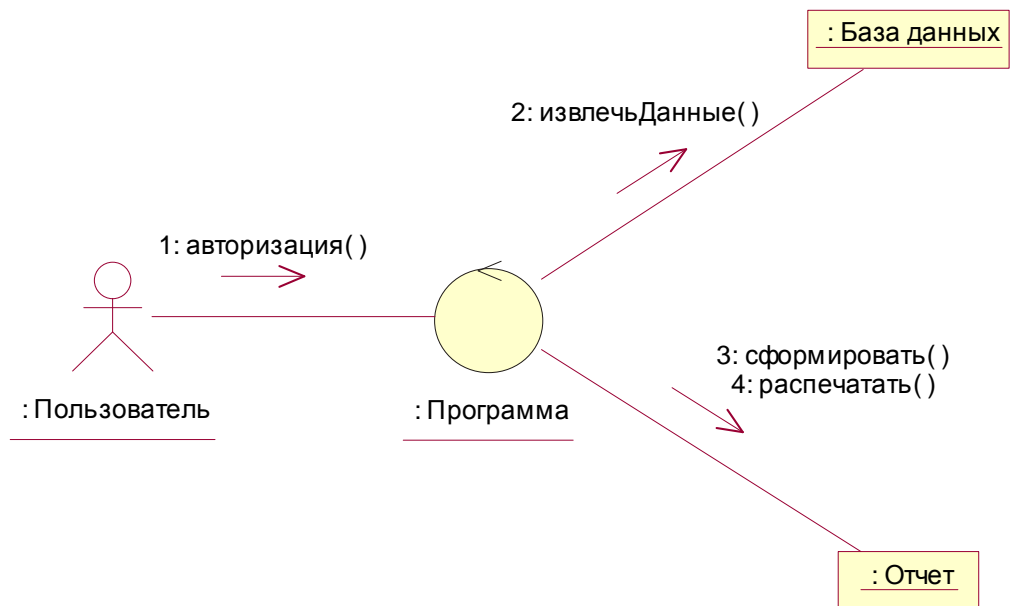


Рисунок 33 – Диаграмма кооперации для модели простейшей информационной системы

2.3.3 Диаграмма последовательности (*sequence diagram*)

Диаграмма последовательности отображает временные особенности передачи и приема сообщений между объектами. С ее помощью можно описать полный контекст взаимодействий как своеобразный «график жизни» всей совокупности объектов, взаимодействующих между собой для реализации варианта использования программной системы, достижения бизнес-цели или выполнения какой-либо задачи.

Каждый объект на этой диаграмме изображается в виде прямоугольника, как и на диаграмме кооперации, однако располагаются эти прямоугольники последовательно слева направо, причем крайним слева изображается объект – инициатор взаимодействия (как правило, это актер). Порядок расположения объектов определяется исключительно соображениями удобства.

Из каждого объекта «вытекает» вертикальная пунктирная линия – его *линия жизни* (object lifeline). Для статических объектов эта линия продолжается до самого низа диаграммы, для динамических – до специального символа уничтожения объекта.

Явное выделение активности объекта отмечается *фокусом управления* (focus of control), который изображается в виде вытянутого узкого прямоугольника, «нанизанного» на линию жизни. Каждый объект за время существования может получать фокус управления сколько угодно раз. Фокус управления актера, как правило, существует в системе постоянно, отмечая характерную активность такого объекта.

Сообщения на диаграмме последовательности имеют такой же смысл и почти такое же описание, как и на диаграмме коопераций (здесь номер сообщения опускается, последовательность определяется слева направо и сверху вниз). Если объект посылает сообщение самому себе, оно называется *рефлексивным*. Если в результате рефлексивного сообщения создается новый процесс, то он изображается в виде *рекурсивного* (вложенного) фокуса управления.

Для изображения ветвления возле каждой ветви в квадратных скобках должно быть указано соответствующее условие в форме булевского выражения.

Все вышеперечисленные элементы диаграммы последовательности представлены на рис. 34.

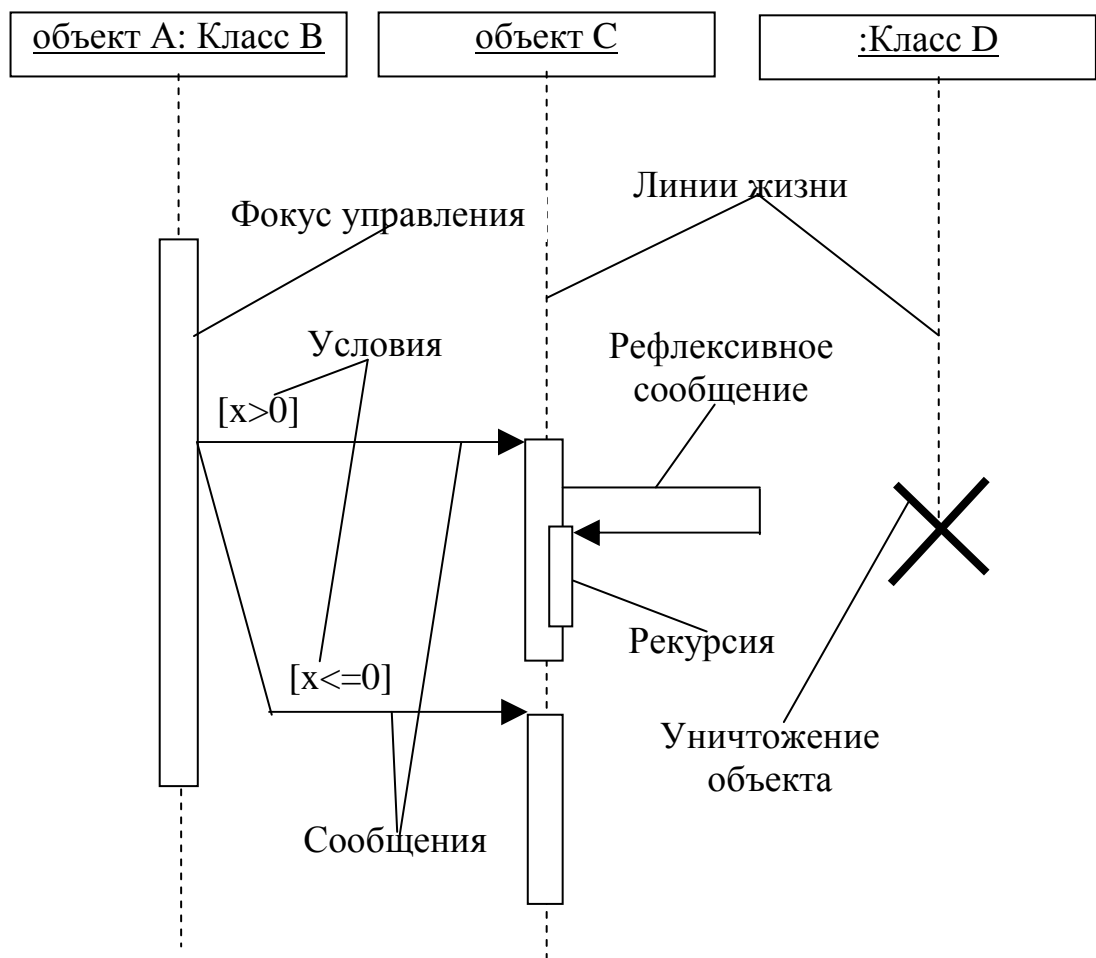


Рисунок 34 – Элементы диаграммы последовательности

На рис. 35 и 36 представлены диаграммы последовательности для модели системы управления банкоматом из [2] и для модели простейшей информационной системы.

Следует отметить, что Rational Rose позволяет не строить диаграмму последовательности «с нуля», а получить ее автоматически из диаграммы кооперации.

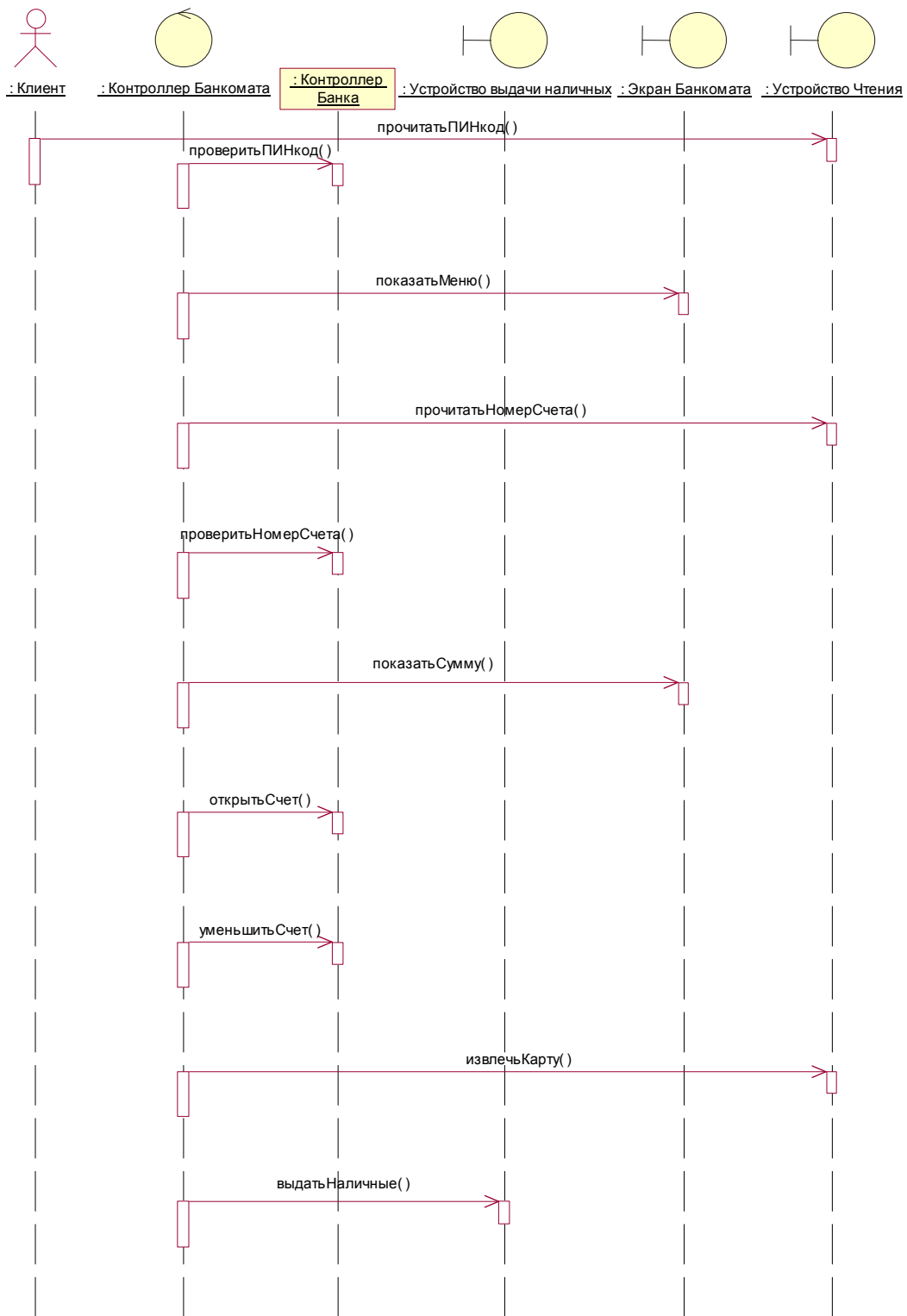


Рисунок 35 – Диаграмма последовательности для модели системы управления банкоматом

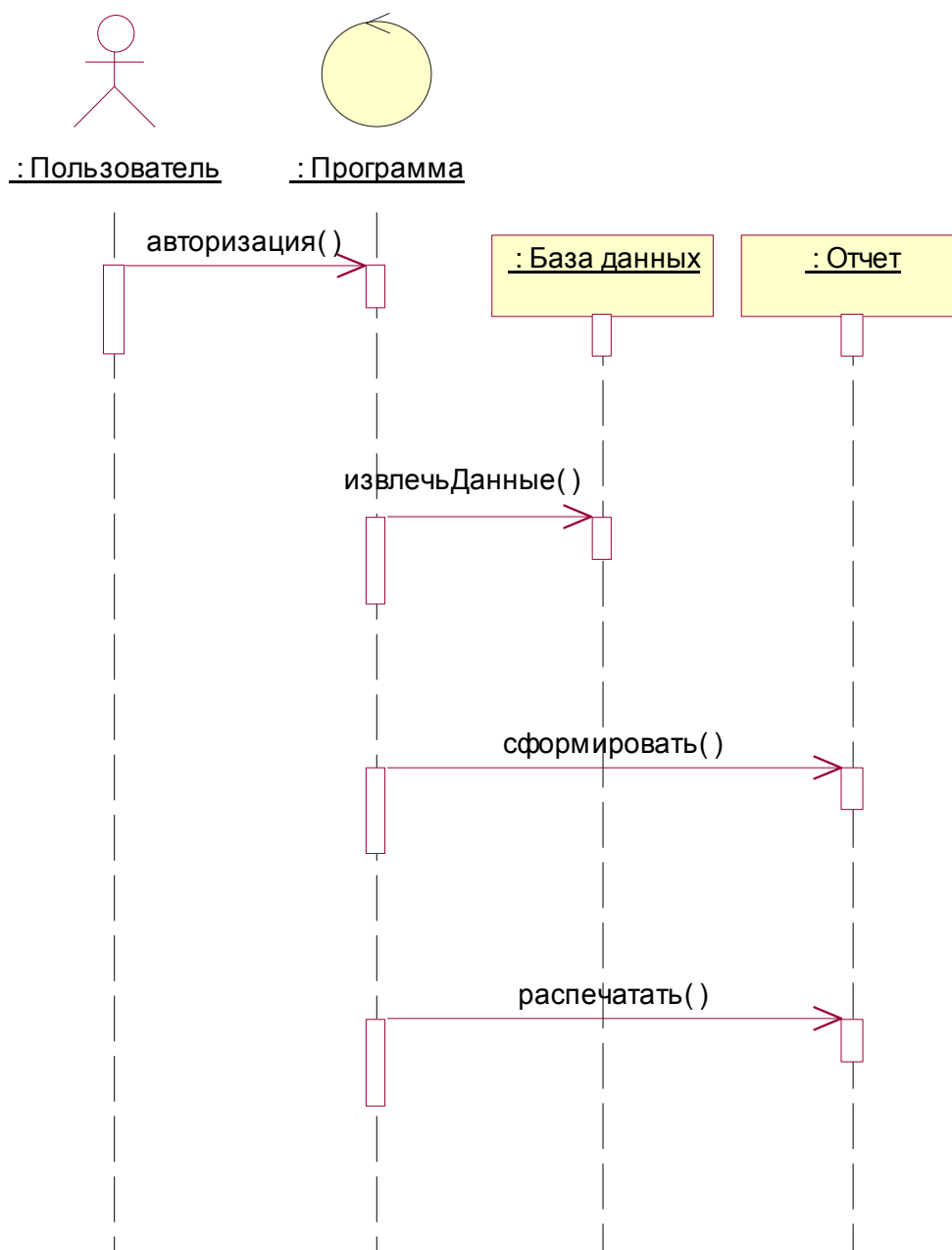


Рисунок 36 – Диаграмма последовательности для модели простейшей информационной системы

2.3.4 Диаграмма состояний (statechart diagram)

Для большинства сложных систем представление динамического взаимодействия элементов модели в виде диаграмм кооперации и последовательности оказывается недостаточным.

В отличие от своих предшественниц, диаграмма состояний описывает процесс изменения состояний системы при реализации всех вариантов использования. При этом такие изменения могут быть вызваны воздействиями со стороны других элементов или извне системы.

Главное назначение данной диаграммы – описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение моделируемой системы (или какой-то подсистемы) в течение всего ее жизненного цикла.

По своей сути диаграмма состояний является графом специального вида, который служит для представления некоторого конечного автомата. Основные понятия – состояние, переход и условия.

Под **состоянием** понимается абстрактный метакласс, используемый для моделирования отдельной ситуации, в течение которой выполняется определенное условие. На диаграмме изображается прямоугольник с закругленными вершинами, который может быть разделен горизонтальной линией (рис. 37).

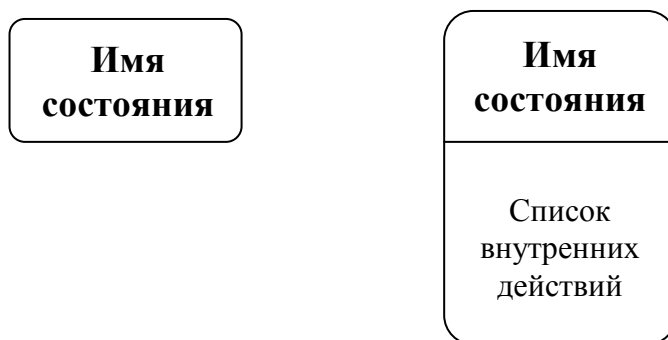


Рисунок 37 – Изображения состояний

Имя состояния представляет собой законченное предложение и записывается с прописной буквы. В исключительных случаях имя может отсутствовать – такое состояние называется *анонимным*.

Список внутренних действий состояния определяет деятельность системы при ее нахождении в данном состоянии. Каждое такое действие записывается отдельной строкой в следующем формате:

метка действия / выражение действия

Метка действия указывает на обстоятельства или условия, при которых будет выполняться это действие, и представляет собой или одно из четырех служебных слов, или идентификатор события, запускающего действие (т.н. *внутренний переход*). Стандартные метки действия приведены ниже:

- *entry* – указывает на входное действие, то есть действие, которое должно быть выполнено в момент входа в состояние;
- *exit* – указывает на выходное действие, то есть действие, которое должно быть выполнено в момент выхода из состояния;
- *do* – определяет некоторую деятельность (*do activity*), которая будет выполняться в течение всего времени нахождения системы в данном состоянии или до выполнения специального условия;
- *include* – обращается к конечному подавтомату, имя которого указывается вместо выражения действия.

Пример состояния со списком внутренних действий – ввод пароля – приведен на рис. 38 [2].

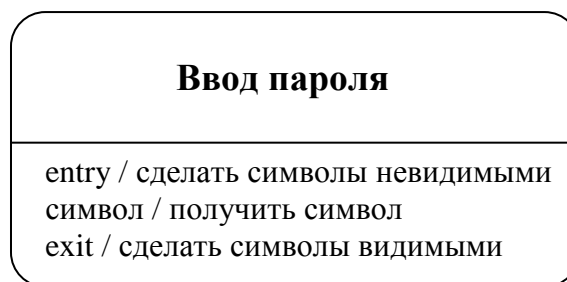


Рисунок 38 – Пример состояния со списком внутренних действий

Начальное состояние (*initial state*) представляет собой частный случай состояния, которое не содержит никаких внутренних действий, и

в котором система (объект) находится в начальный момент. Графически изображается в виде закрашенного кружка (рис. 39, а).



Рисунок 39 – Начальное и конечное состояния

Конечное состояние (final state) представляет собой частный случай состояния, которое также не содержит никаких внутренних действий и в котором моделируемая система (объект) находится после завершения работы. Графически изображается в виде закрашенного кружка, помещенного в окружность (рис. 39, б).

Простой переход (simple transition) представляет собой отношение между двумя последовательными состояниями, которое указывает на факт смены одного состояния (*источника*) другим (*целевым*). Переход осуществляется при наступлении события – завершении деятельности в исходном состоянии (*нетриггерный переход*) или поступлении сообщения (*триггерный переход*), а также при удовлетворении так называемого *сторожевого условия*. Изображается сплошной линией со стрелкой, направленной от исходного состояния в целевое, сопровождающейся строкой текста. Если исходное и целевое состояния совпадают, переход называют *переходом в себя* и изображают петлей со стрелкой. *Сторожевое условие* (guard condition) записывается в квадратных скобках после (или вместо) строки-события и представляет собой некоторое булевское выражение, то есть должно принимать одно из двух значений – «истина» или «ложь».

Составное состояние (composite state) состоит нескольких вложенных *подсостояний* (substate). В зависимости от взаимного расположения подсостояния могут быть последовательными (sequential substates) или параллельными.

На рис. 40 показано составное состояние с двумя вложенными последовательными подсостояниями на примере работы телефонного ап-

парата (прецедент «дозвон до абонента») [2], на рис. 41 – составное состояние с двумя вложенными параллельными подсостояниями.

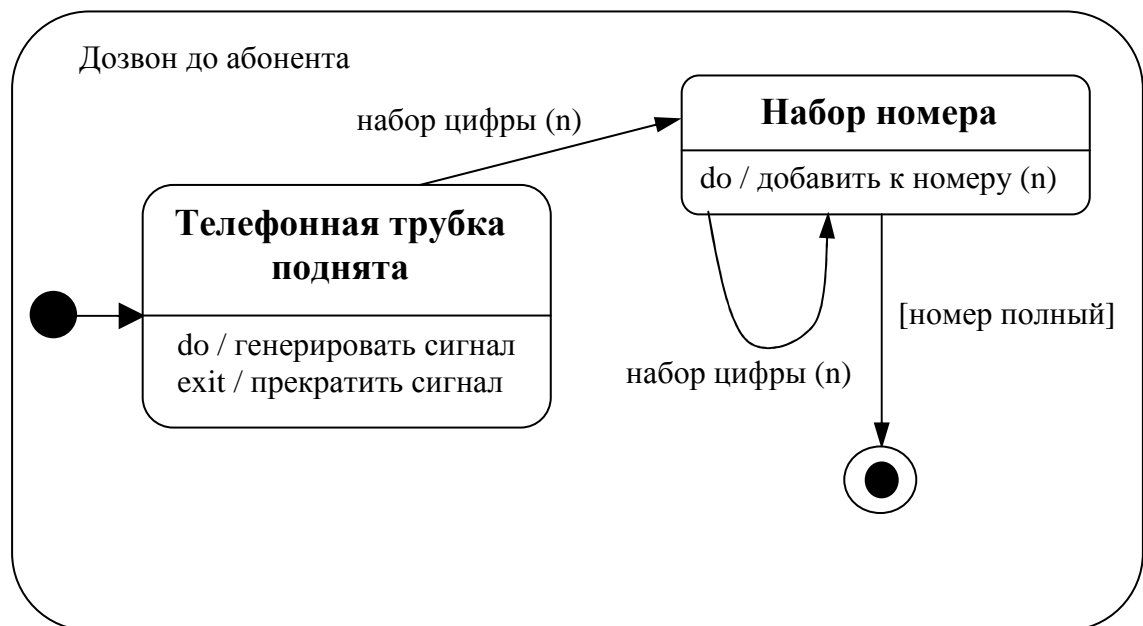


Рисунок 40 – Диаграмма состояний дозвона до абонента

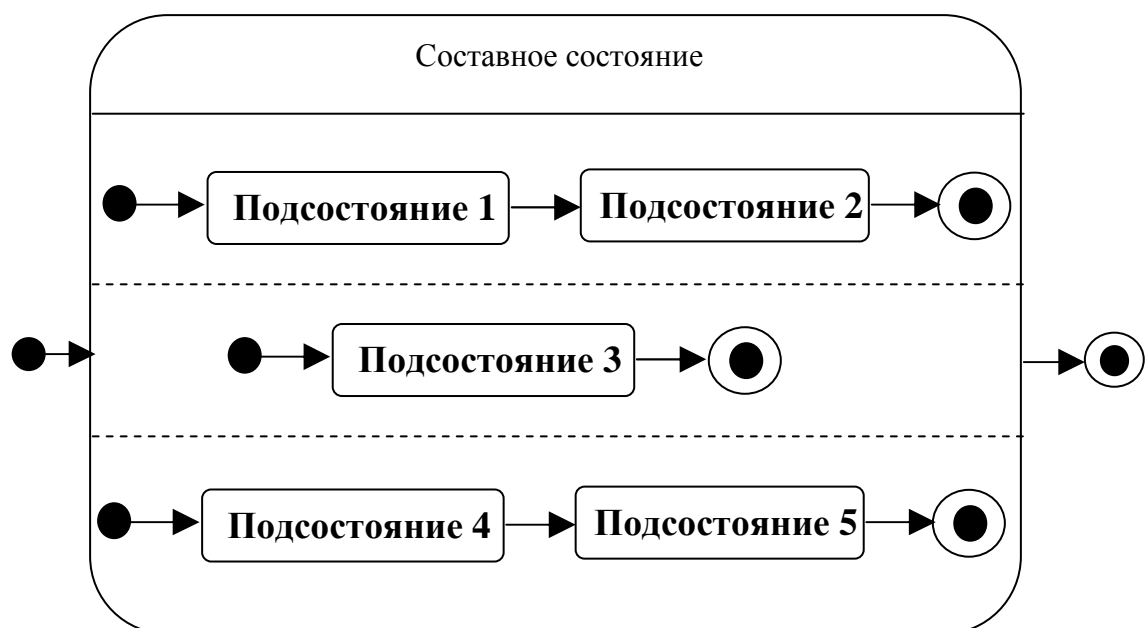


Рисунок 41 – Составное состояние с параллельными подсостояниями

Если при выходе из подсостояния необходимо его «запомнить» для последующего возвращения, используются так называемые *исторические состояния* (history state). *Неглубокое историческое состояние* (shallow history state) изображается буквой Н (рис. 42, а) и определяет первое

подсостояние при входе в составное состояние (при первом входе оно автоматически заменит начальное состояние). *Глубокое историческое состояние* (deep shallow history state) изображается символами H^* и служит для запоминания всех вложенных подсостояний (рис. 42, б).

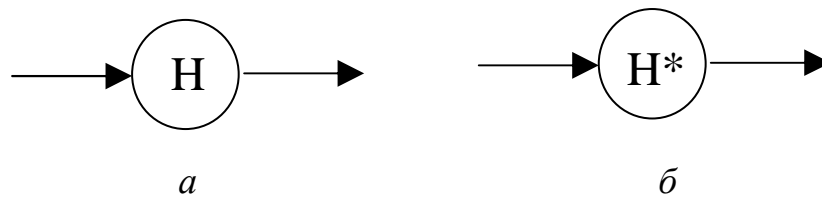


Рисунок 42 – Недавнее (неглубокое) и давнее (глубокое) исторические состояния

Если нужно явно показать, что некоторый переход может иметь несколько исходных состояний или несколько целевых состояний, он называется *параллельным переходом* и изображается вертикальной чертой. На рис. 43 показаны две разновидности параллельного перехода – *слияние* (join) и *разделение* (fork).

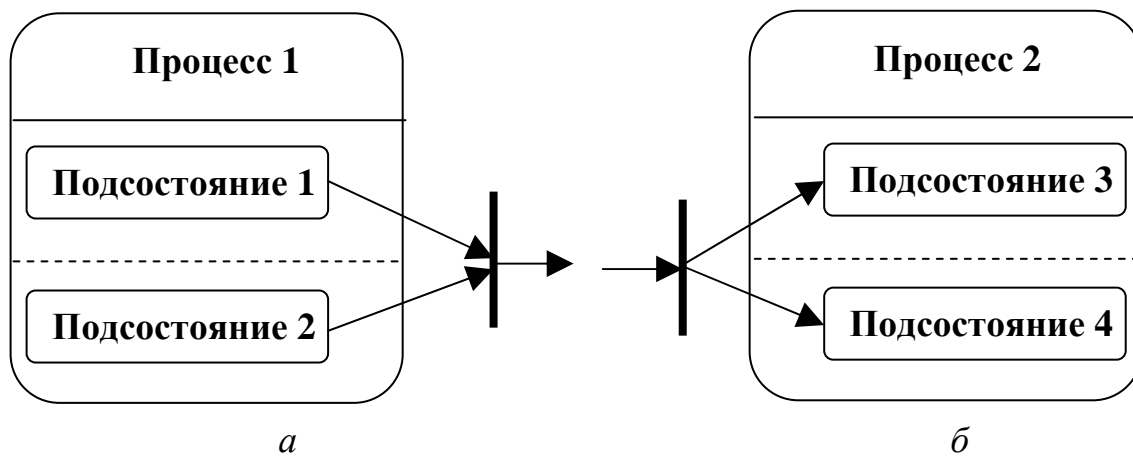


Рисунок 43 – Параллельные переходы вида «слияние» (а) и «разделение» (б)

Синхронизирующее состояние (synch state) используется совместно с переходом-слиянием или переходом-разделением для явного указания влияния событий в одном подавтомате на другой подавтомат и обозначается небольшой окружностью со «звездочкой» внутри.

На рис. 44 приведен пример использования синхронизирующих состояний при моделировании строительства дома [2], на рис. 45 изо-

бражена диаграмма состояний для модели системы управления банкоматом, а на рис. 46 – возможный вариант диаграммы состояний для работы простейшей информационной системы.

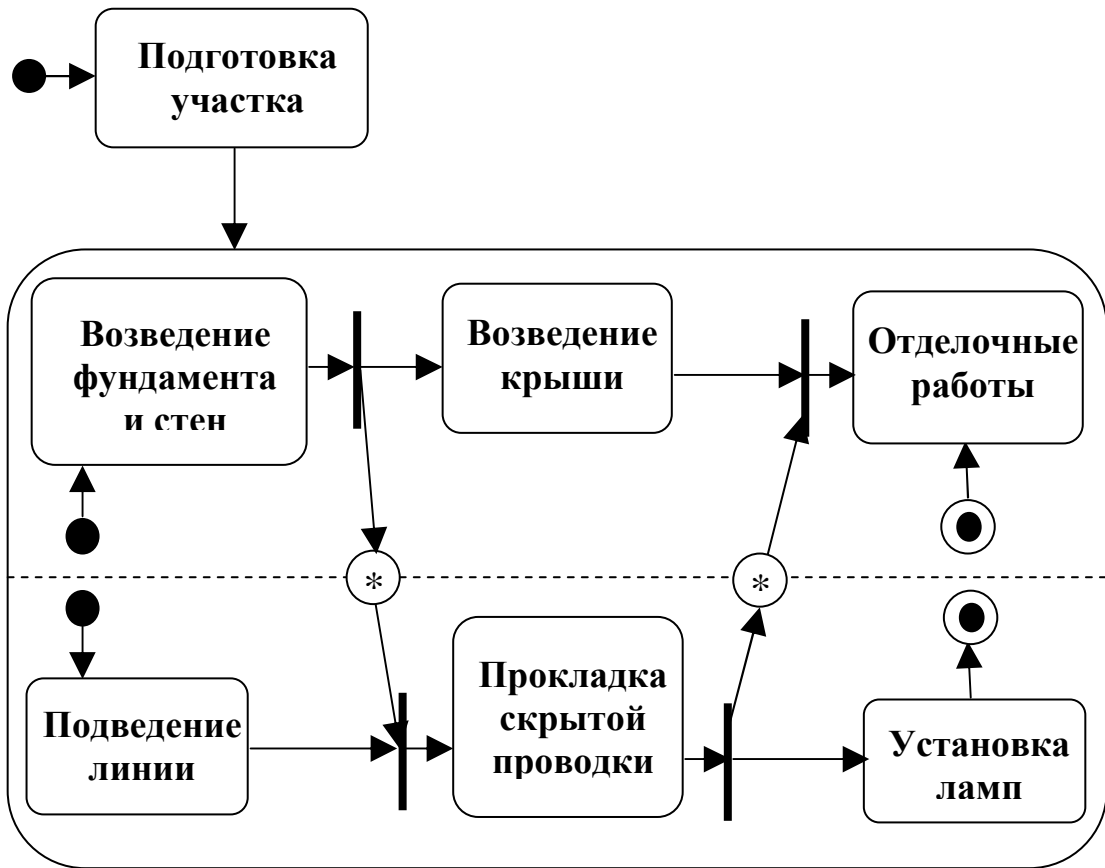


Рисунок 44 – Диаграмма состояний моделирования строительства дома

2.3.5 Диаграмма деятельности (activity diagram)

С увеличением сложности системы усиливается важность строгого соблюдения последовательности выполняемых действий. Для моделирования процесса выполнения операций в языке UML используются диаграмма деятельности, которая по своей сути является частным случаем диаграммы состояний.

Деятельность (activity) представляет собой некоторую совокупность отдельных вычислений, которые могут приводить к некоторому результату или действию.

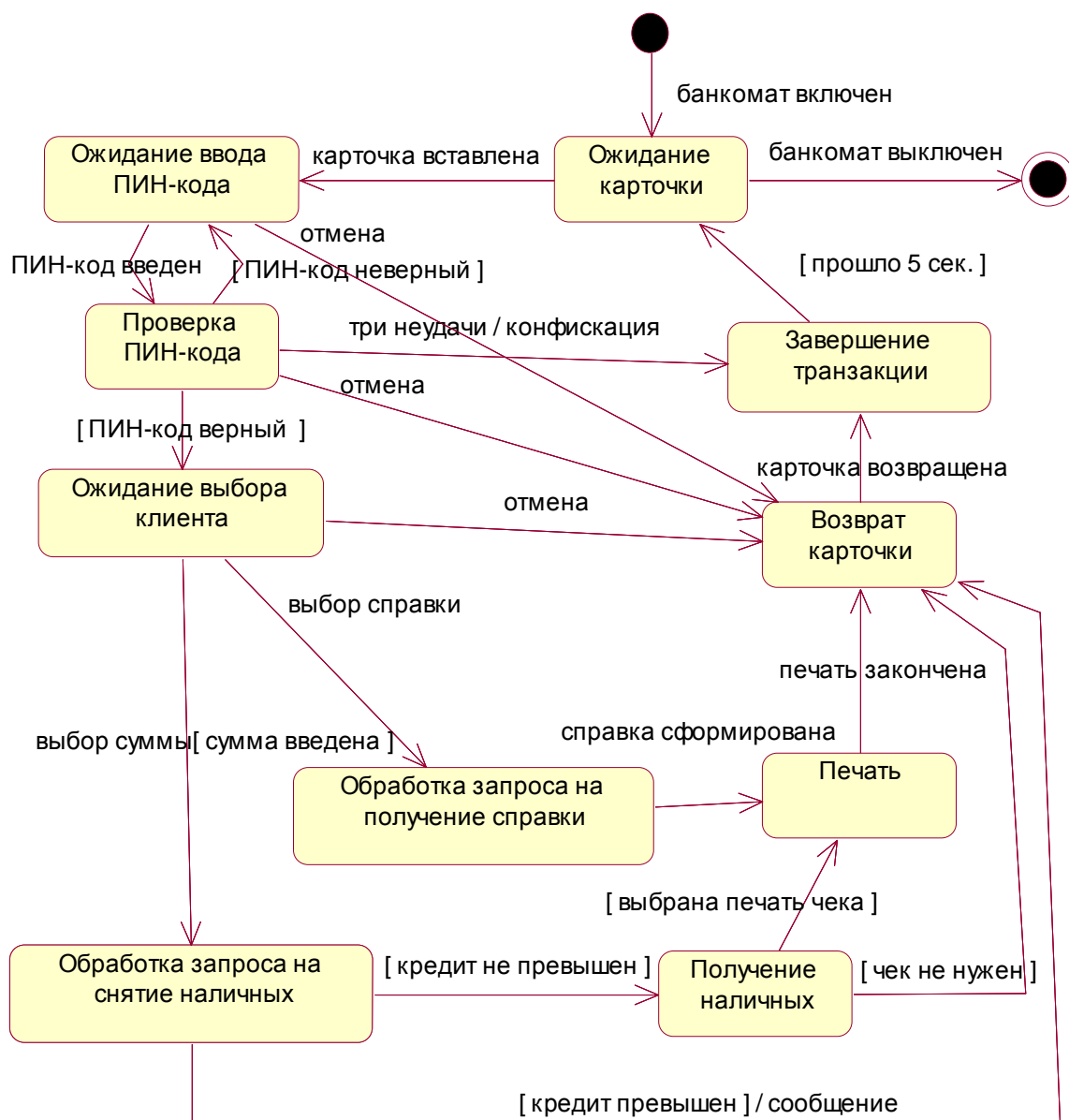


Рисунок 45 – Диаграмма состояний для модели системы управления банкоматом

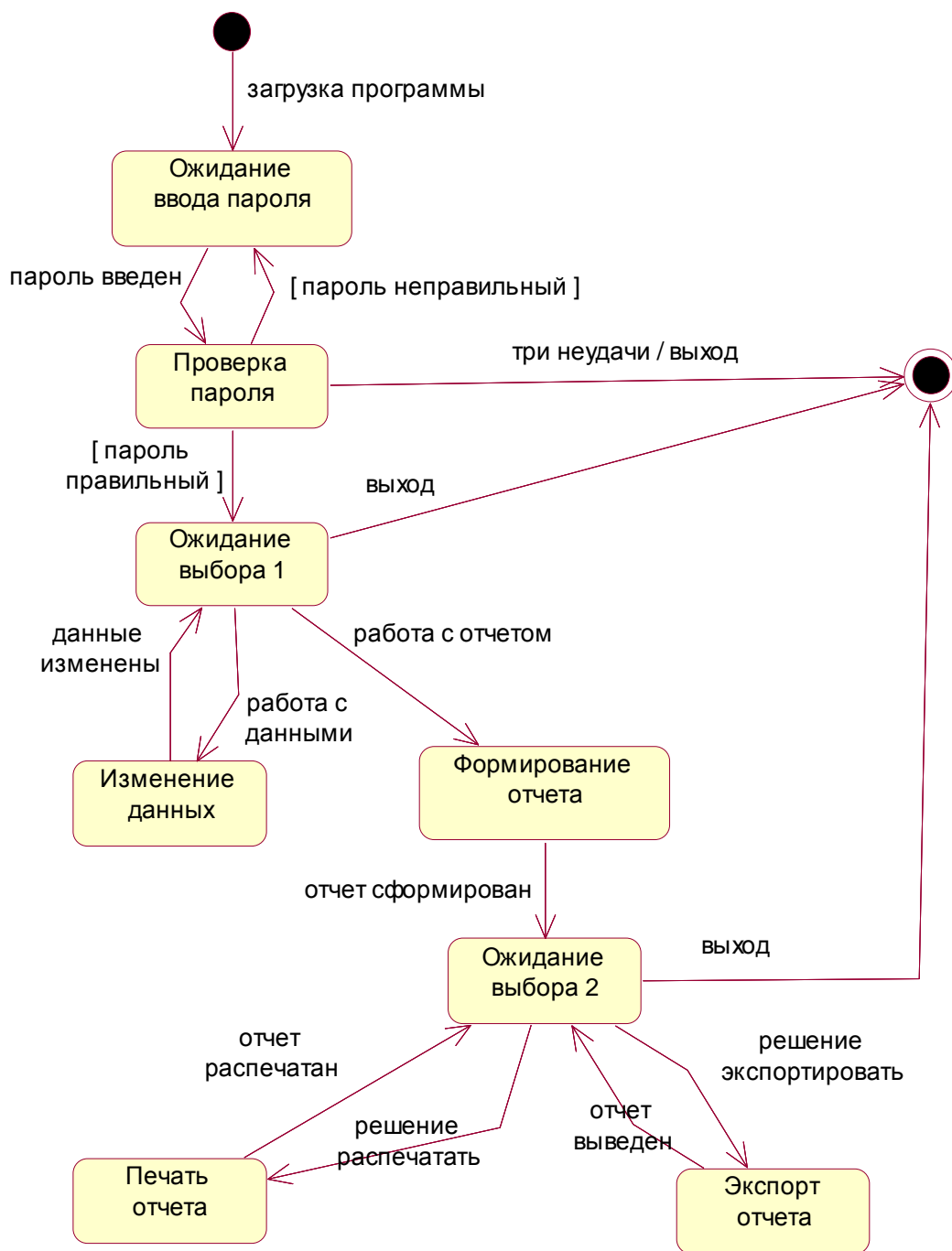
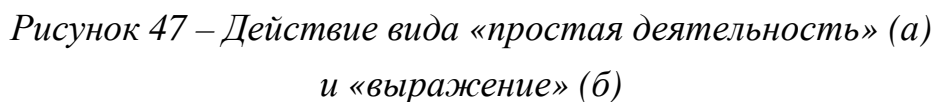


Рисунок 46 – Диаграмма состояний для модели простейшей информационной системы

Состояние действия (action state) является специальным случаем состояния с некоторым входным действием и, как минимум, одним выходящим из состояния переходом, который неявно предполагает, что входное действие уже завершилось. Состояние действия не может иметь

Графически состояние действия изображается прямоугольником со сферическими сторонами, внутри которого записывается имя состояния-действия в форме *выражения-действия* (action-expression), как это показано на рис. 47.



Ветвление изображается символом *решения* (decision) – небольшого ромба без текста. Объединение альтернативных ветвей осуществляется при помощи такого же ромба, но называемого уже *соединением* (merge).

58

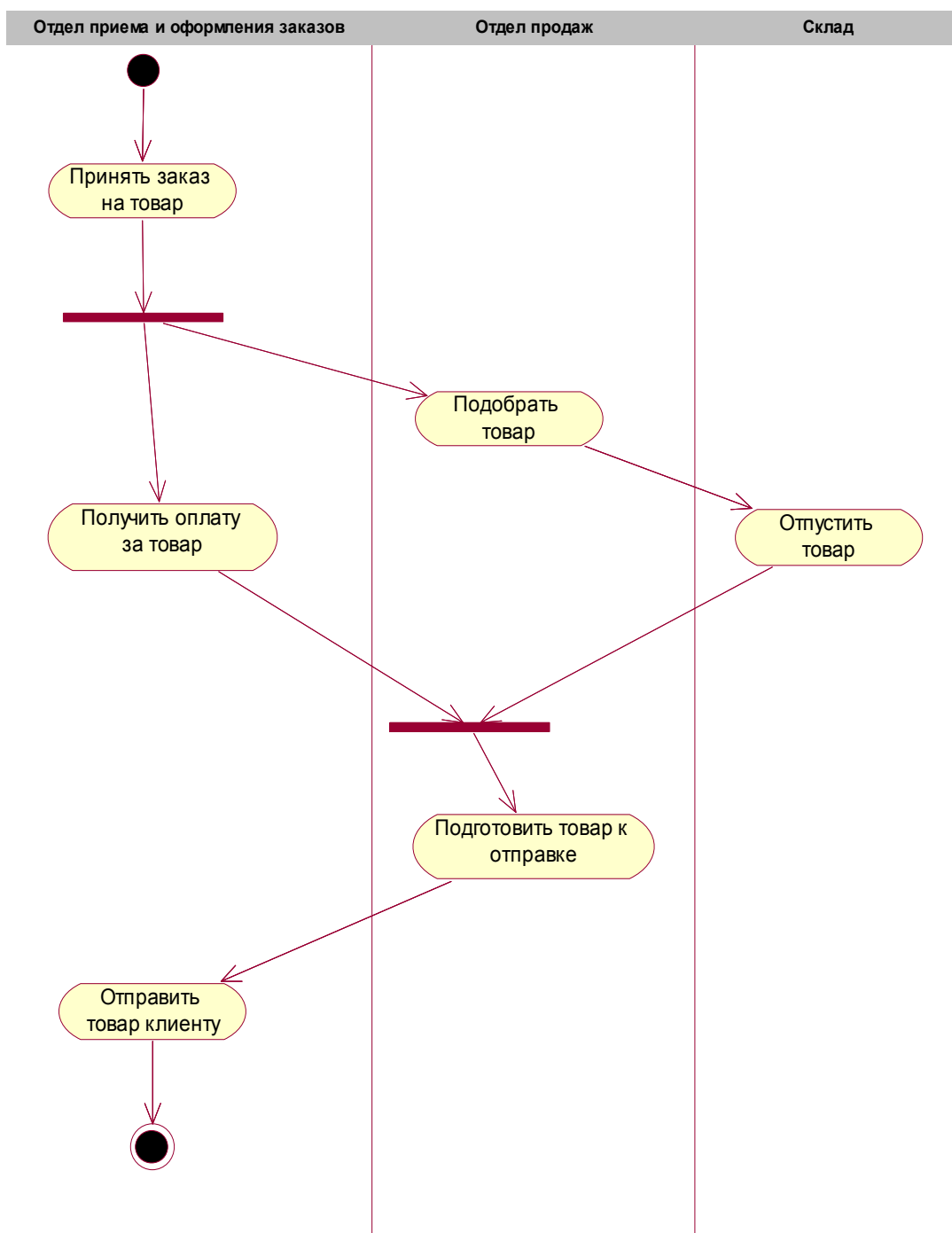


Рисунок 48 – Диаграмма деятельности для торговой компании

На рис. 49 построена диаграмма деятельности для нашего сквозного примера – процесса функционирования системы управления банкоматом [2], а на рис. 50 – диаграмма деятельности для простейшей информационной системы.

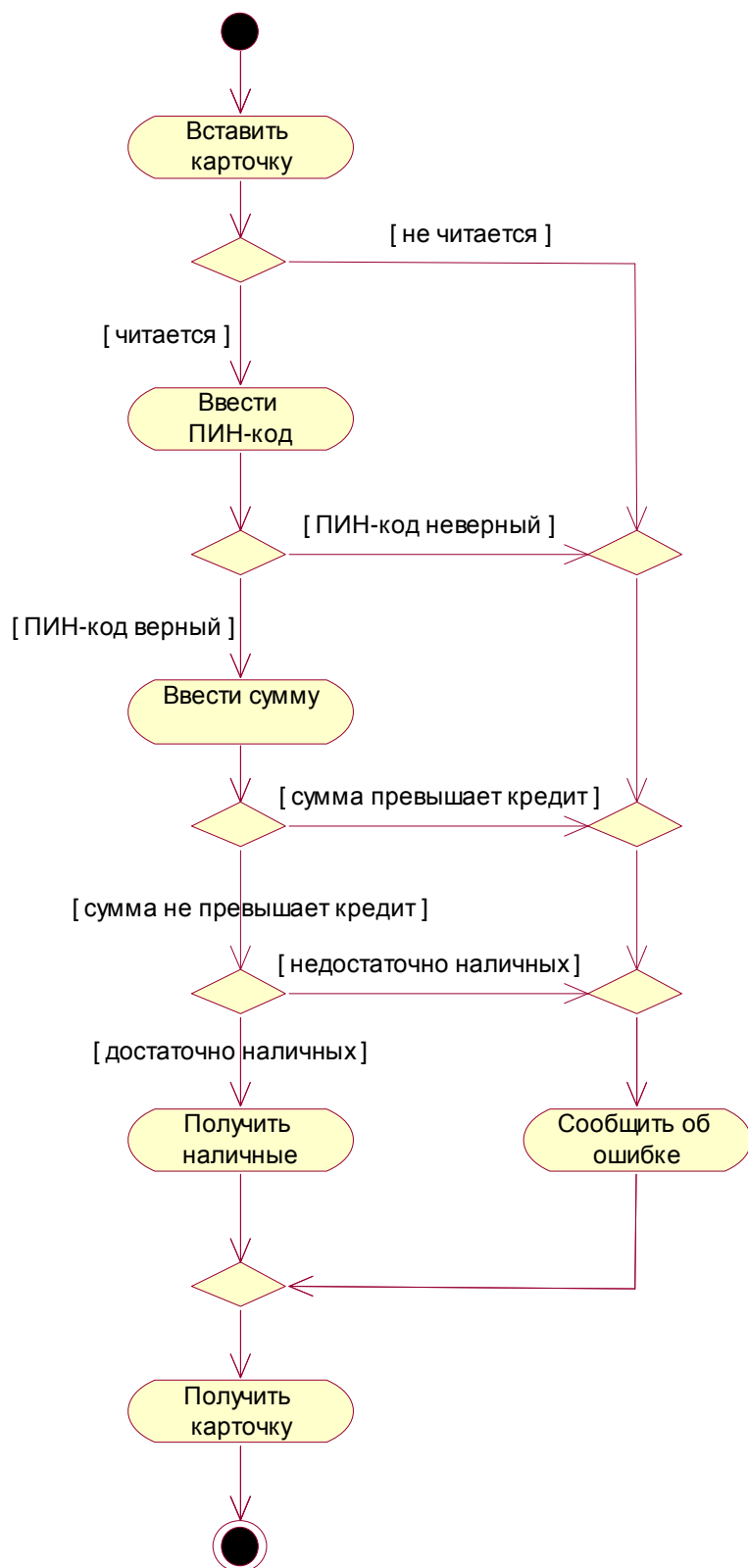


Рисунок 49 – Диаграмма деятельности для процесса функционирования системы управления банкоматом

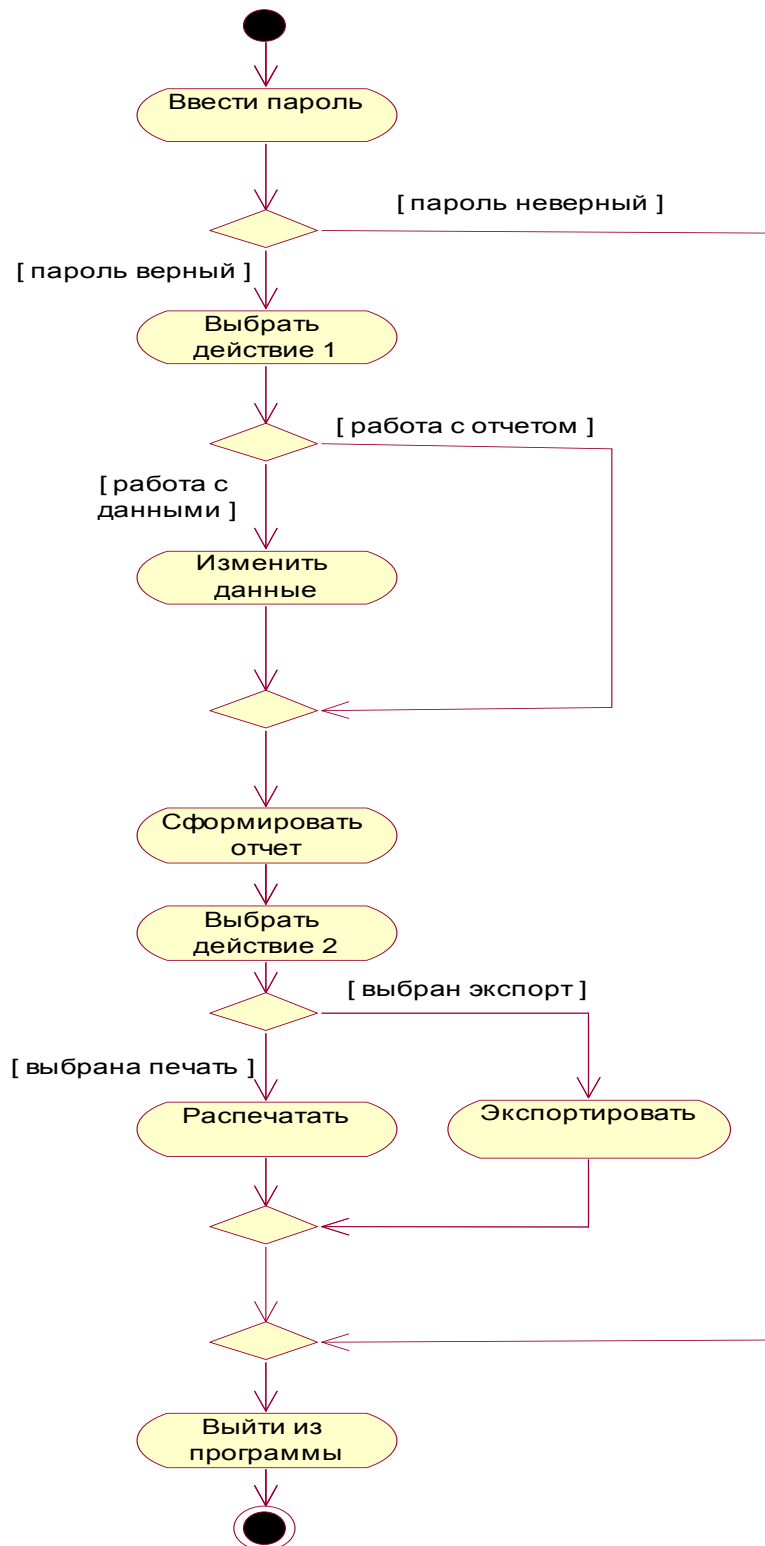


Рисунок 50 – Диаграмма деятельности для простейшей информационной системы

2.4 Диаграммы физического моделирования

Для создания конкретной физической системы необходимо некоторым образом реализовать все элементы логического представления в конкретные материальные сущности. Моделирование такого преобразования описывают так называемые *диаграммы реализации* (implementation diagram), к которым относятся *диаграмма компонентов* и *диаграмма развертывания*.

2.4.1 Диаграмма компонентов (component diagram)

Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный и исполняемый код (как правило, компонент соответствует файлу операционной системы). Основными графическими элементами этой диаграммы являются компоненты, интерфейсы и зависимости между ними.

Компонент (component) представляет некоторую физическую сущность и может реализовывать некоторый набор интерфейсов. Графически изображается прямоугольником с именем и со вставленными слева двумя прямоугольниками поменьше (рис. 51).

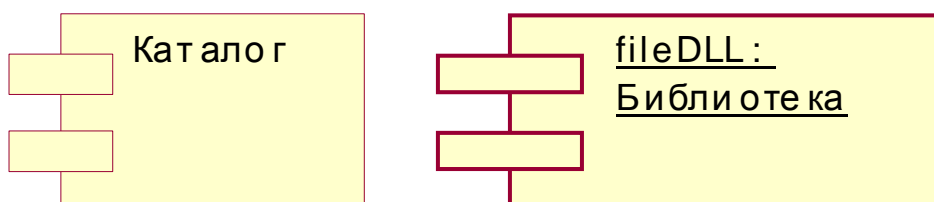


Рисунок 51 – Изображение компонентов

Запись имени компонента зависит от того, представлен он в качестве типа или в качестве экземпляра. В первом случае записывается имя типа с заглавной буквы, во втором – в форме «имя компонента : имя типа».

В качестве собственных имен принято использовать имена исполняемых файлов (EXE), динамических библиотек (DLL), web-страниц (HTML), текстовых файлов (TXT или DOC), файлов справки (HLP), баз

данных (DB или DBF) или файлов с исходными текстами программ (PAS, CPP, JAVA, PL и т.п.). Внешний вид таких компонентов на диаграмме не определен нотацией языка UML и зависит от среды построения диаграмм (CASE-средства). Представление некоторых компонентов в среде IBM Rational Rose показано на рис. 52.

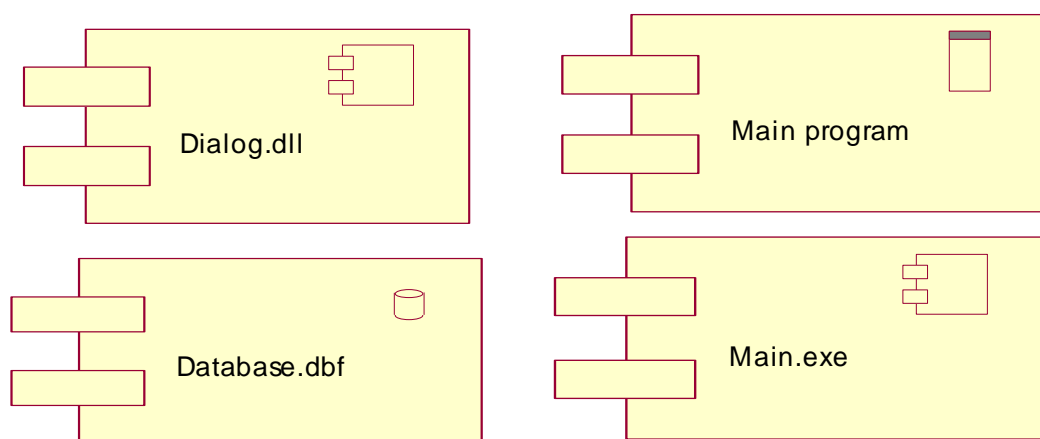


Рисунок 52 – Варианты изображения компонентов в среде IBM RR

Интерфейс (interface) изображается окружностью, которая соединяется с компонентом отрезком линии без стрелок, или в виде прямоугольника класса со стереотипом <<interface>> (рис. 53).

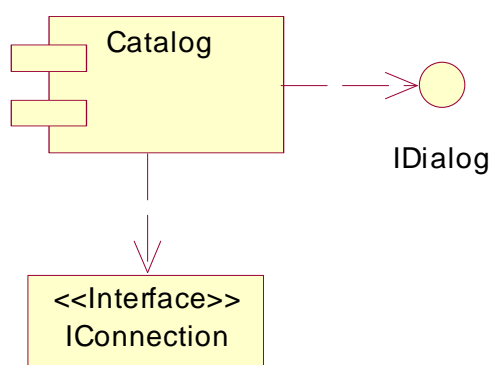


Рисунок 53 – Изображение интерфейсов

Отношение зависимости (dependency) изображается пунктирной линией со стрелкой, направленной от клиента (зависимого элемента) к источнику (независимому элементу). На рис. 53 было показано, что компонент зависит от своих интерфейсов.

Подробнее об особенностях построения диаграммы компонентов и о дополнительных возможностях можно прочитать в [2-7]. На рис. 54 представлена диаграмма компонентов системы управления банкоматом [2], на рис. 55 – один из возможных вариантов диаграммы компонентов простой информационной системы (предполагается реализация в среде Borland Delphi).

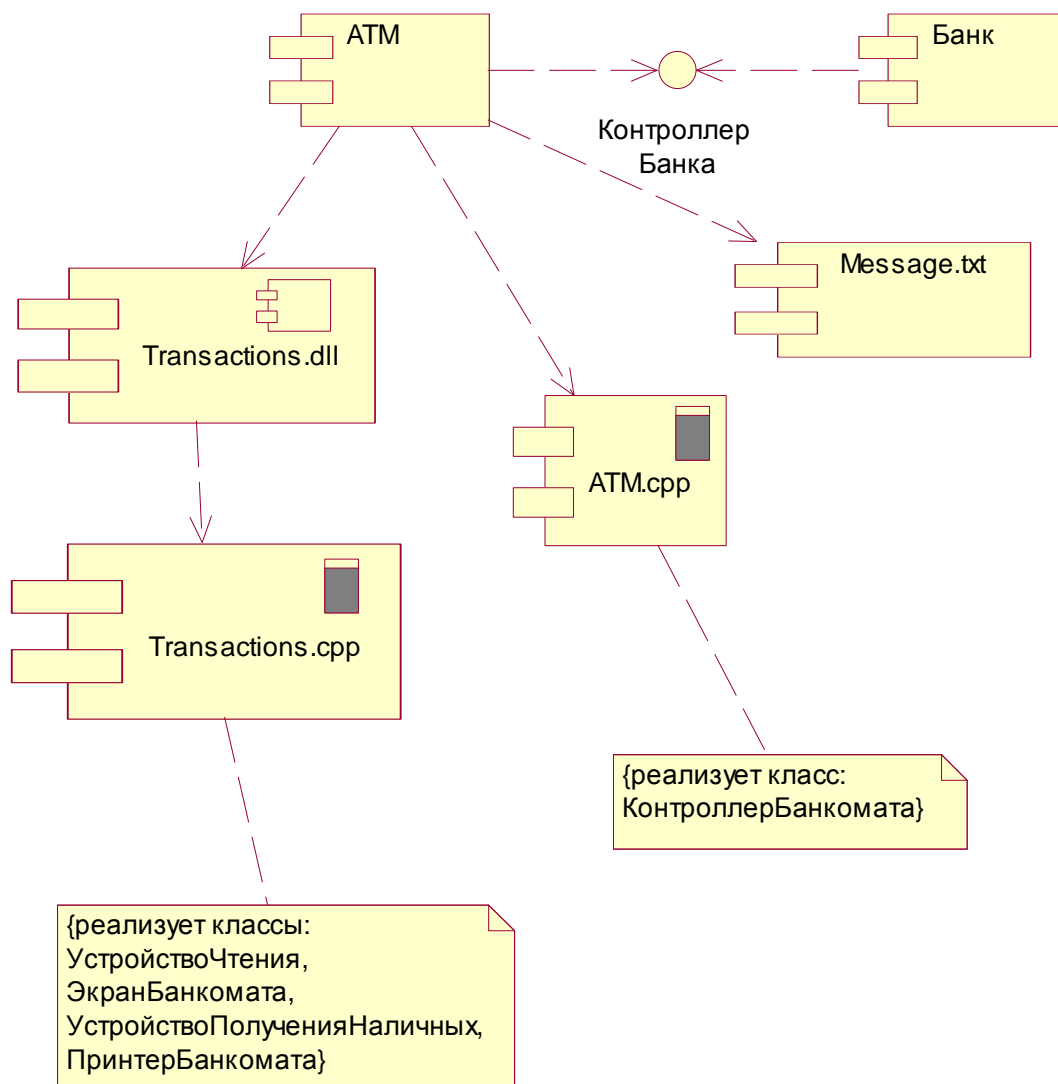


Рисунок 54 – Диаграмма компонентов системы управления банкоматом

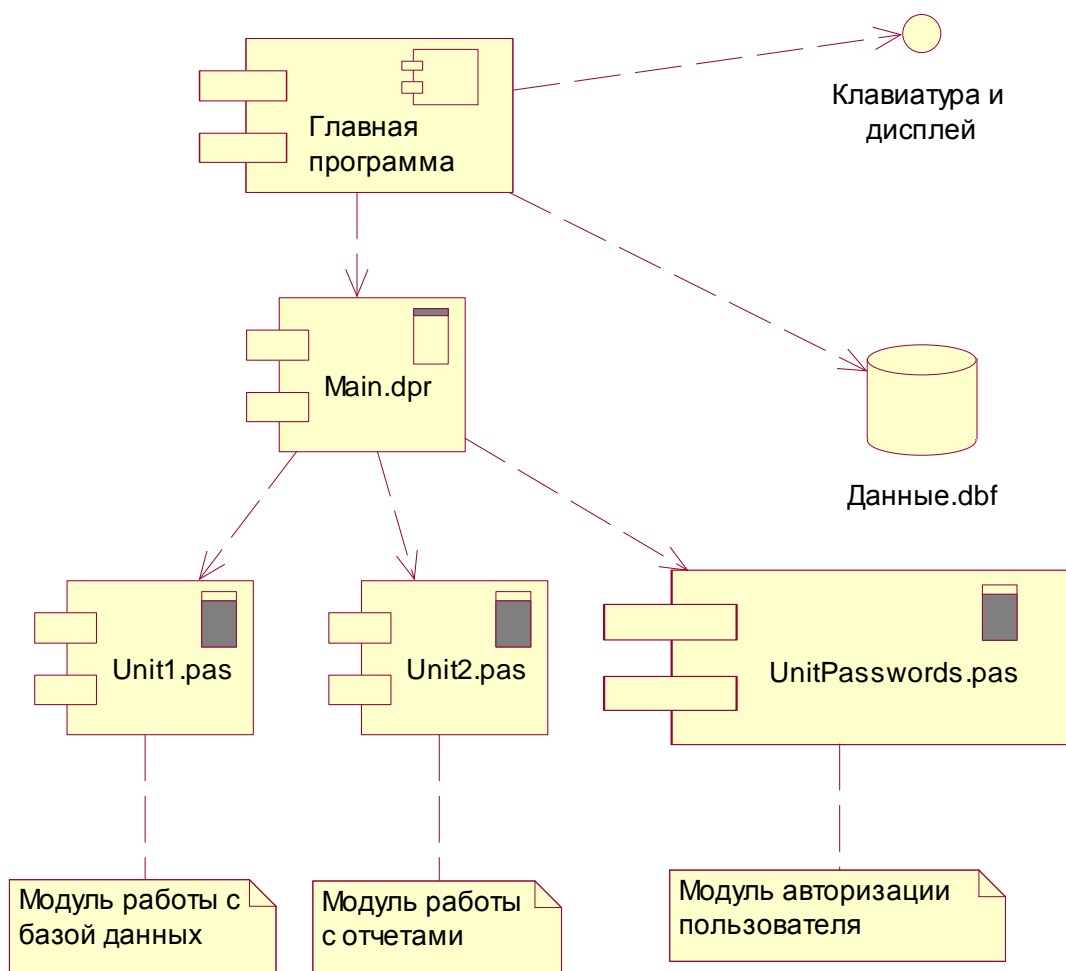


Рисунок 55 – Диаграмма компонентов информационной системы

2.4.2 Диаграмма развертывания (deployment diagram)

Если разрабатываемая программа предполагает «локальную» работу на одном компьютере, то построением диаграммы компонентов проектирование системы и заканчивается. Однако сложные программные системы часто реализуются в сетевом варианте, предполагающем использование разных вычислительных платформ и технологий доступа к данным. Для представления общей конфигурации и топологии распределенной программной системы, а также маршрутов передачи информации между аппаратными устройствами применяется диаграмма развертывания.

Основные элементы этой диаграммы – процессоры и устройства (узлы), процессы и связи между ними.

Узел (node) представляет собой некоторый физически существующий элемент системы, который может обладать вычислительным ресурсом (один или несколько процессоров, оперативная память и т. п.). К узлам относятся компьютеры, принтеры, модемы, сканеры и другие подобные устройства. Графически узел изображается в форме параллелепипеда с именем, которое может быть как именем типа узла, так и именем узла-экземпляра. Ресурсоемкий узел (processor) изображается в форме параллелепипеда с закрашенными боковыми гранями (рис. 56).

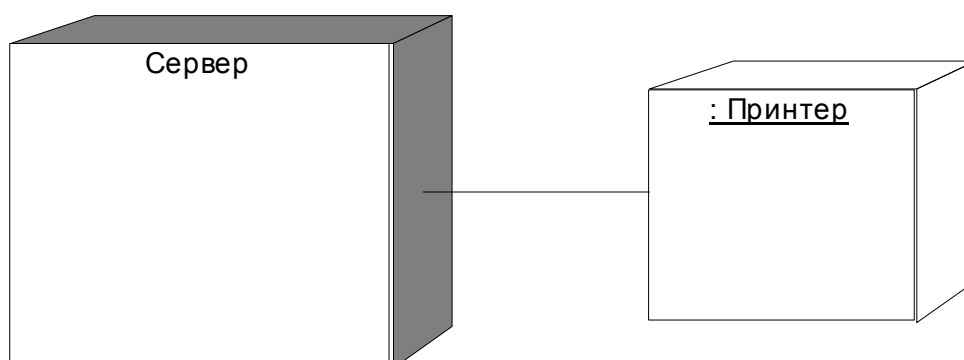


Рисунок 56 – Узлы на диаграмме развертывания

В качестве отношений между узлами выступают физические соединения между ними, а также зависимости между узлами и другими компонентами. **Соединения** являются разновидностью ассоциации и изображаются отрезками прямой линии без стрелок (см. рис. 56). **Зависимости** отображаются отрезками пунктирной линии со стрелками, направленными от узла к зависимым от него компонентам.

На рис. 57 представлена диаграмма развертывания системы управления банкоматом [2], на рис. 58 – простейший вариант диаграммы развертывания информационной системы в случае ее клиент-серверной реализации.

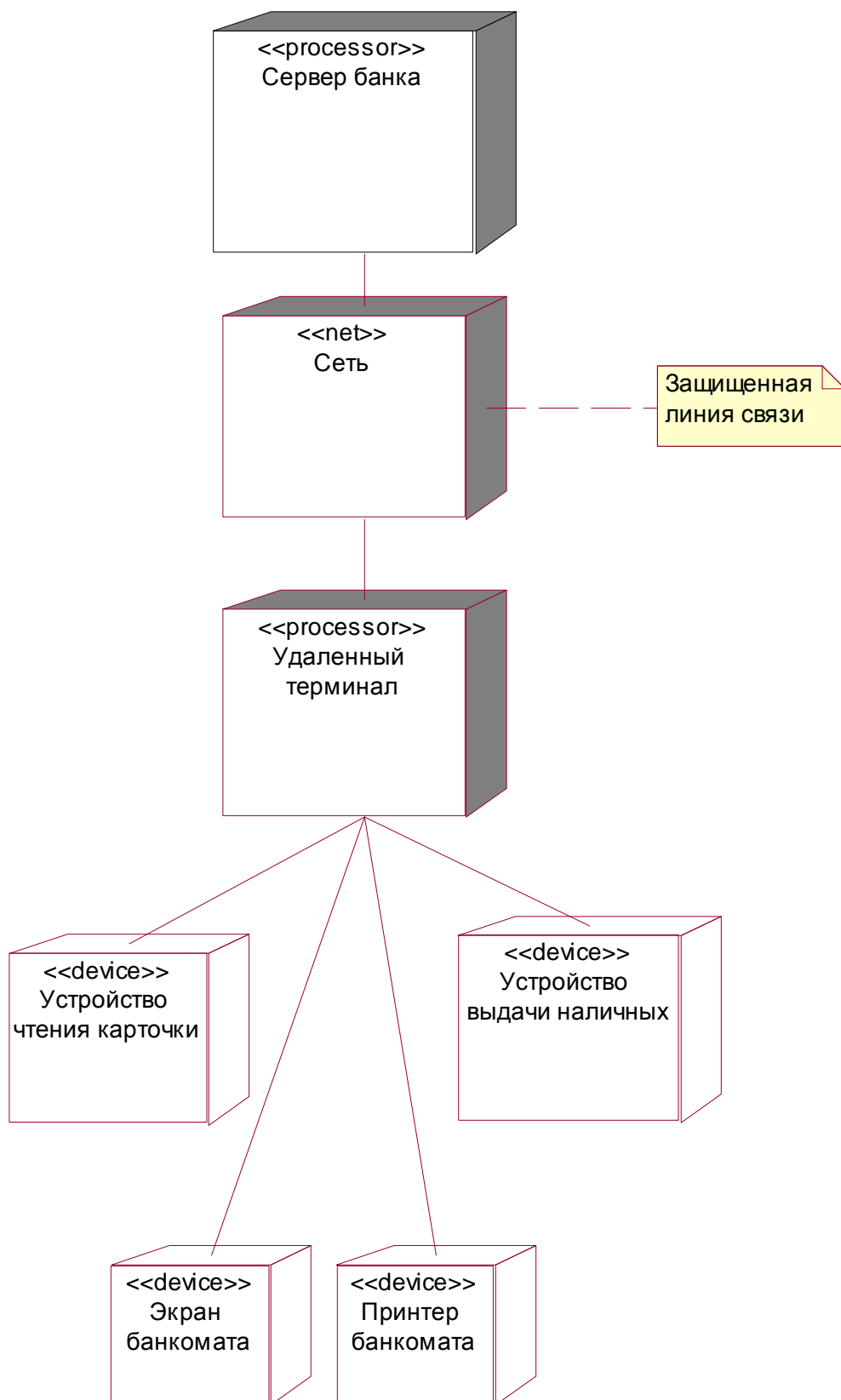


Рисунок 57 – Диаграмма развертывания системы управления банкоматом

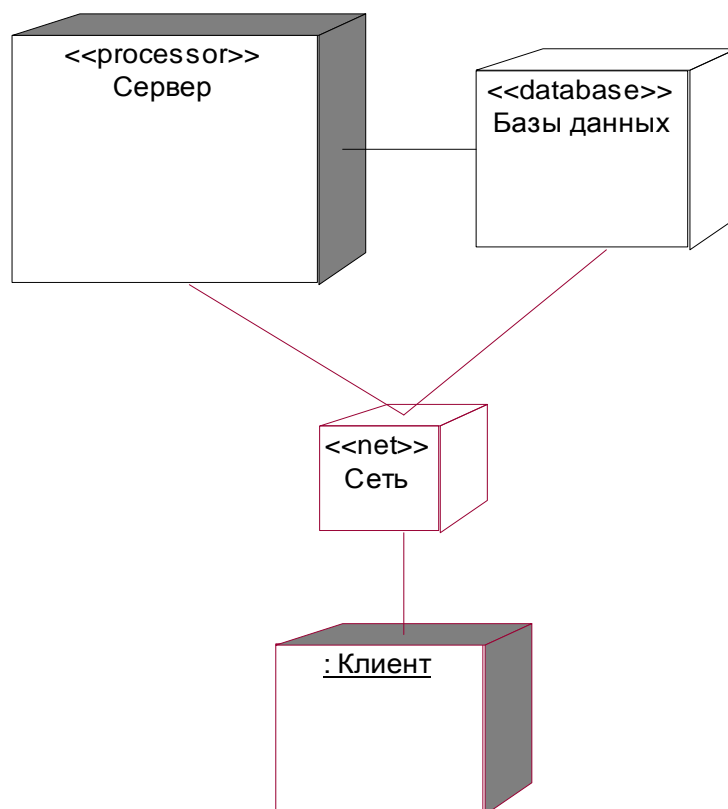


Рисунок 58 – Диаграмма развертывания информационной системы

3 ПРОЕКТИРОВАНИЕ ПРОГРАММНЫХ СИСТЕМ С ИСПОЛЬЗОВАНИЕМ CASE-СРЕДСТВА IBM RATIONAL ROSE

Проектировать систему на языке UML можно разными способами – в том числе «вручную», то есть рисовать диаграммы на листе бумаги или в среде текстового процессора. Понятно, что последний способ нецелесообразен из-за невозможности быстрой корректировки, копирования и проверки на наличие ошибок; обычно используются так называемые CASE-средства (Computer Aided Software/System Engineering – проектирование программ/систем при помощи компьютера).

В настоящее время существует семь наиболее распространенных средств проектирования [8] – IBM Rational Rose, Borland Together, Microsoft Visio, Sparx-Systems Enterprise Architect, GentleWare Poseidon, SmartDraw и Dia. Каждое из них имеет свою особенность (интеграция с MS-Office, удобство использования, простота изучения и т. п.), но фак-

тически стандартом является IBM Rational Rose. Единственным недостатком этого мощного пакета является достаточная сложность его освоения. Уменьшить негативное влияние этого недостатка призван данный раздел учебного пособия.

3.1 Общая характеристика инструментария IBM Rational Rose

CASE-средство IBM Rational Rose позволяет построить все канонические UML-диаграммы в рамках единой модели, проверить модель на наличие ошибок и осуществить экспорт в виде кодов программ.

Спроектированная модель сохраняется в файле с расширением MDL, резервные копии – в файлах с расширением MD~. Одновременно можно работать только с одной моделью – при загрузке новой предыдущая автоматически закрывается.

Работа начинается с выбора будущей среды реализации (рис. 59). Если среда пока точно не определена, рекомендуется выбрать «Rational unified process».

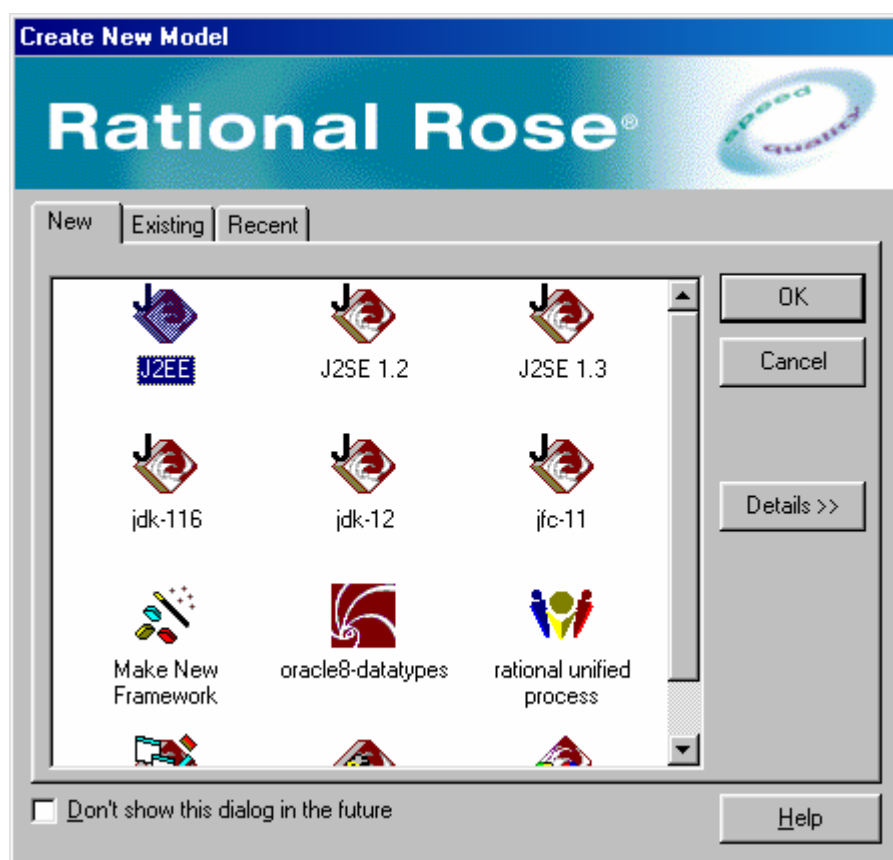


Рисунок 59 – Окно выбора среды реализации

Интерфейс IBM Rational Rose оформлен по аналогии с интерфейсами большинства Windows-приложений, поэтому нет смысла останавливаться на пунктах главного меню и подробном перечислении содержания панели инструментов (рис. 60).

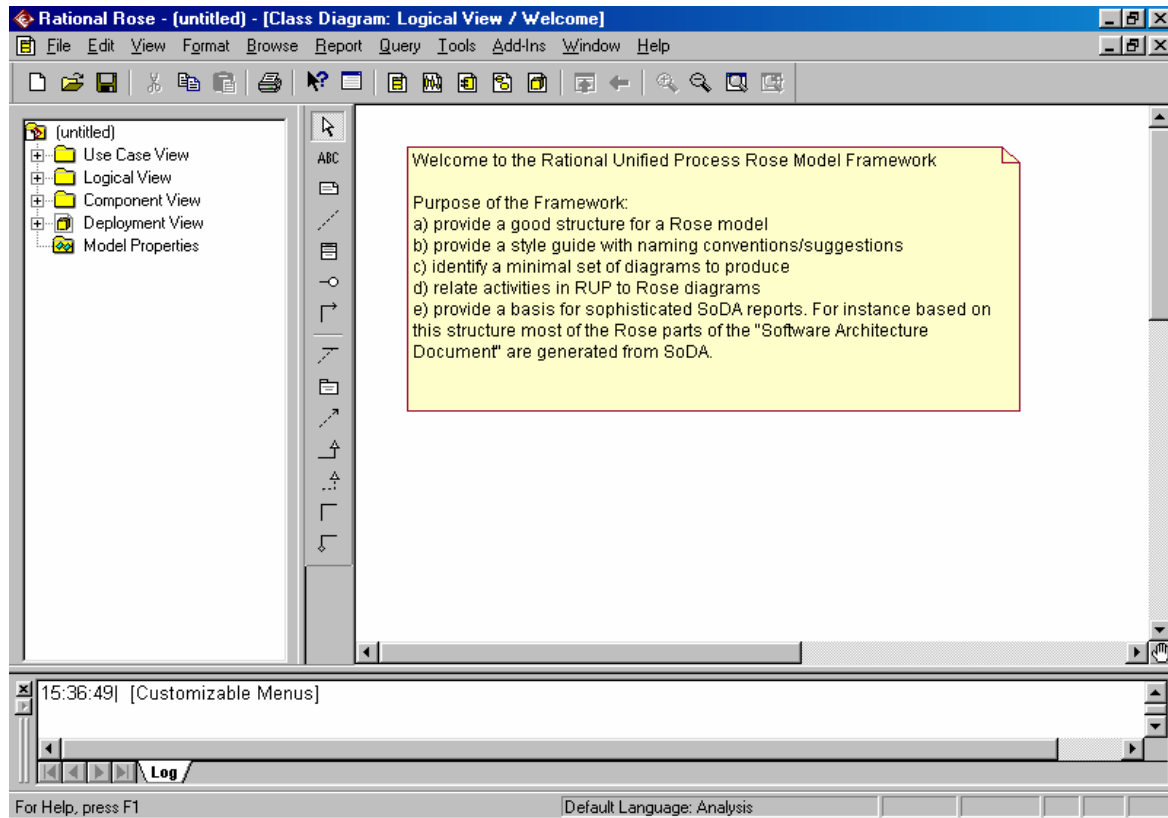


Рисунок 60 – Рабочий интерфейс среды IBM Rational Rose

В левой части экрана располагается **окно браузера проекта**, в котором можно видеть проектируемую систему в виде иерархической структуры, верхними уровнями которой являются «Концептуальное представление» (use case view), «Логическое представление» (logical view), «Компонентное представление» (component view) и «Представление развертывания» (deployment view).

В правой части экрана располагается **окно диаграммы**, где, собственно, и происходит процесс проектирования. Между окном браузера и окном диаграммы располагается **специальная панель инструментов**, содержание которой зависит от выбранной диаграммы. Состав этой панели можно изменять (пункт Customize контекстного меню).

Внизу экрана находится окно журнала, куда выводится служебная информация о выполненных действиях.

Переключение между диаграммами осуществляется либо нажатием соответствующего значка на панели инструментов, либо выбором из главного меню (Browse).

Остальные особенности работы в среде IBM Rational Rose будут понятны в дальнейшем при рассмотрении примера разработки модели простейшей информационной системы.

3.2 Пример разработки модели информационной системы в среде IBM Rational Rose

Согласно RUP (рациональному унифицированному процессу) проектирование системы должно начинаться с построения концептуальной модели – то есть с диаграммы вариантов использования.

Выберем в главном меню пункт «Browse / Use Case Diagram» (или выберем слева «Use Case View / Main») – на экране появится новая диаграмма вариантов использования с размещенными по умолчанию пакетами (рис. 61).

После удаления пакетов поместим на диаграмму актера (размещение компонентов осуществляется стандартным приемом «выдели и щелкни на поле»), при этом сразу введем его имя (в нашем примере – Admin) – рис. 62. Аналогично поместим второго актера (User).

Наша модель предполагает два основных варианта использования – «Ввод и модификация данных» и «Работа с данными» (то есть их извлечение и анализ). Помещение варианта использования происходит подобно помещению актера (рис. 63).

Первый вариант использования предлагается администратору, второй – пользователю, поэтому свяжем их ассоциациями (рис. 64).

Поскольку предполагается два пользователя (Admin и User), системе необходимо их предварительно идентифицировать. Для этого поместим еще один вариант использования «Аутентификация», связав его с двумя остальными отношением зависимости типа «Включение» (рис. 65).

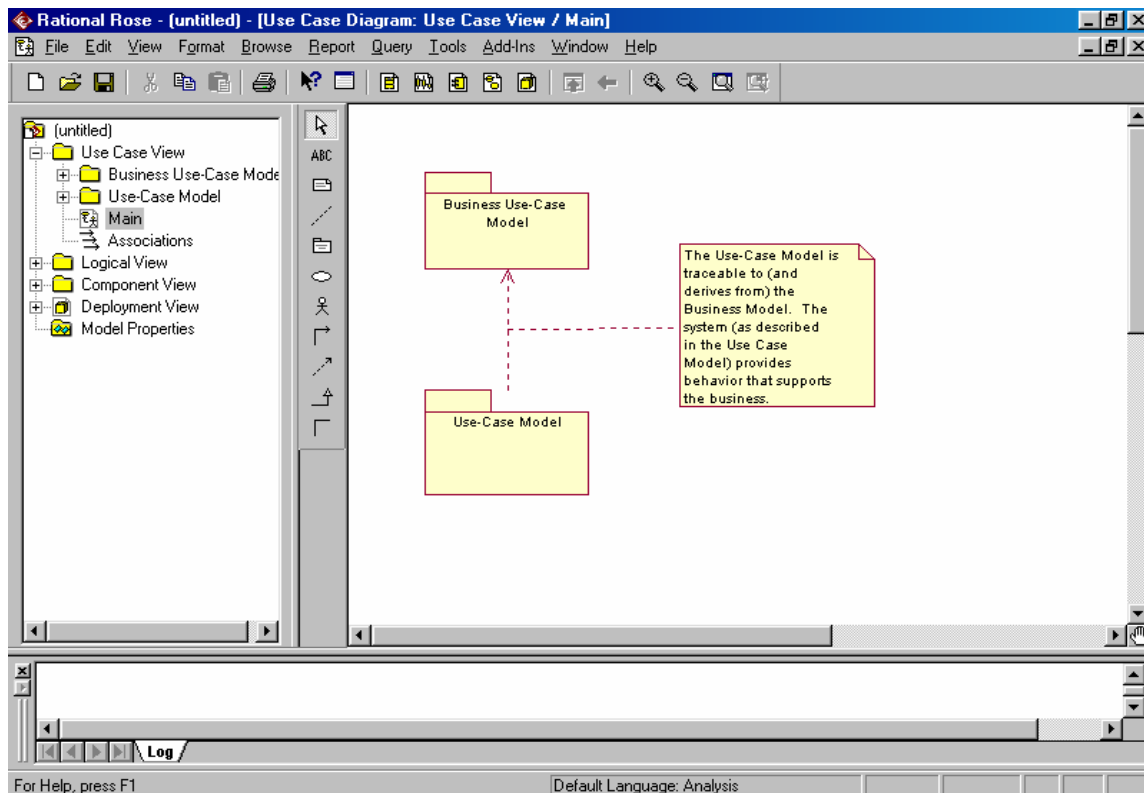


Рисунок 61 – Начало построения диаграммы вариантов использования

Вид зависимости определяется двойным щелчком мыши и выбором «Stereotype» в появившемся окне (рис. 66).

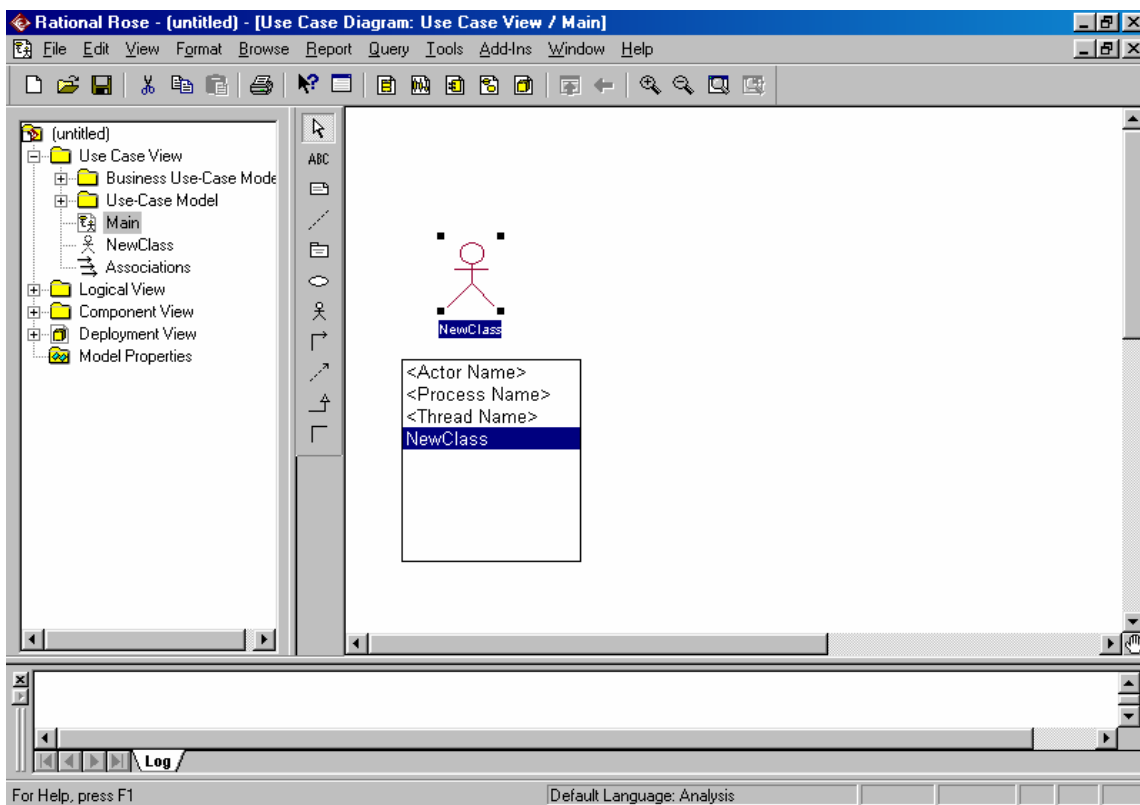


Рисунок 62 – Помещение актера на диаграмму вариантов использования

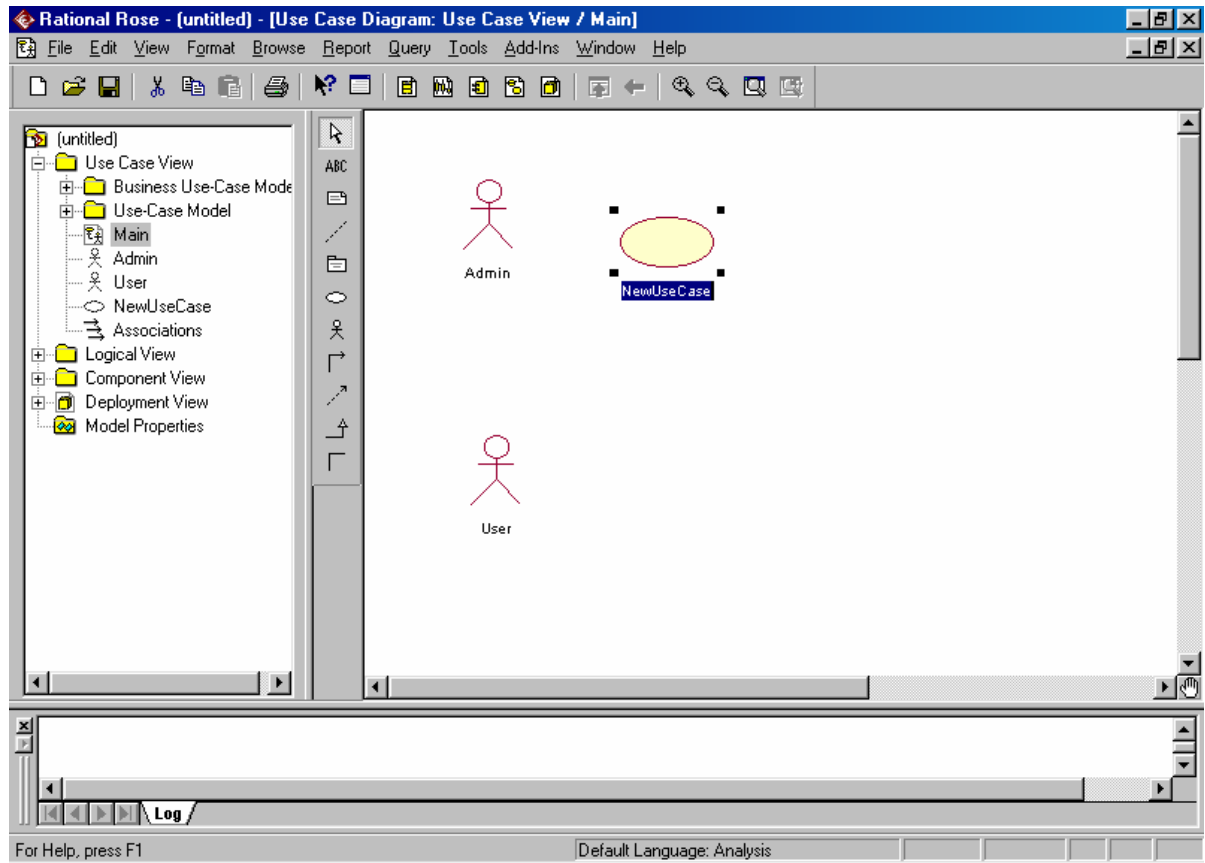


Рисунок 63 – Помещение варианта использования на диаграмму

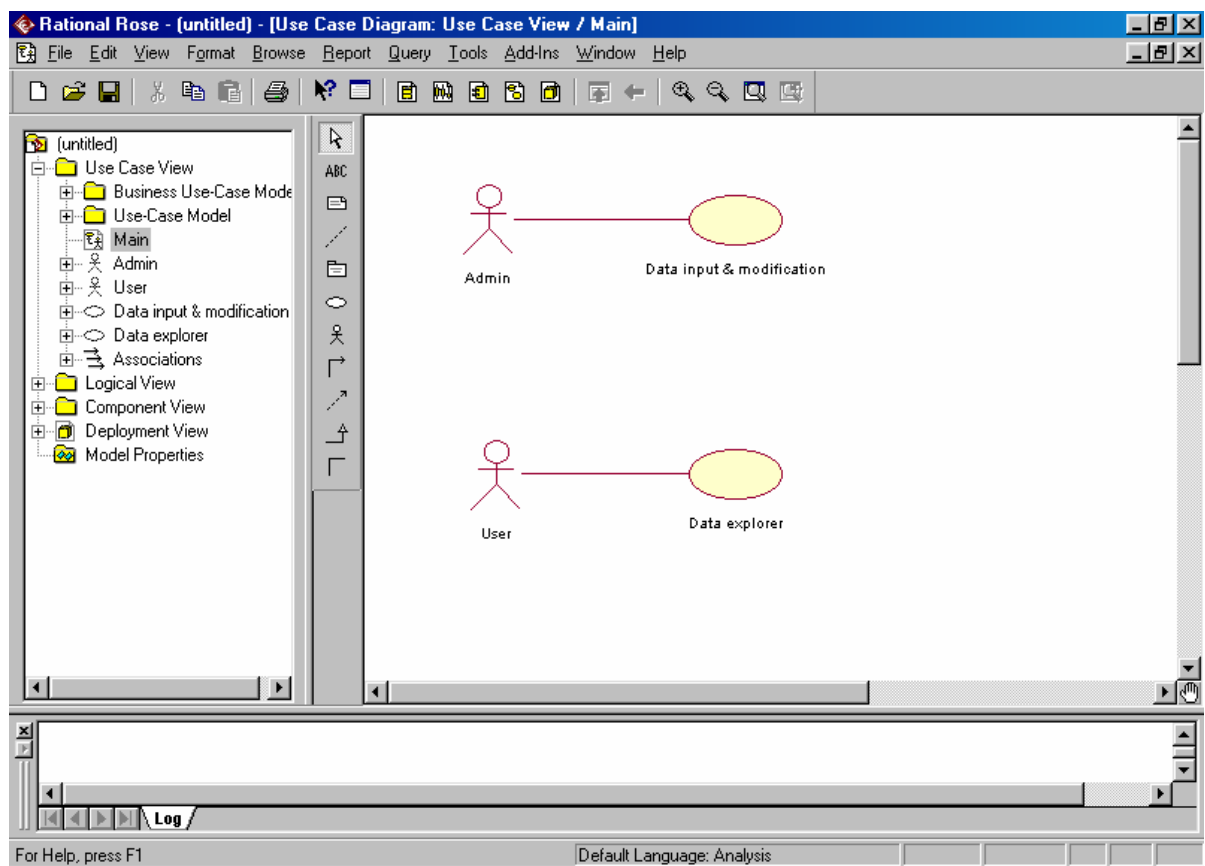


Рисунок 64 – Добавление связей между компонентами

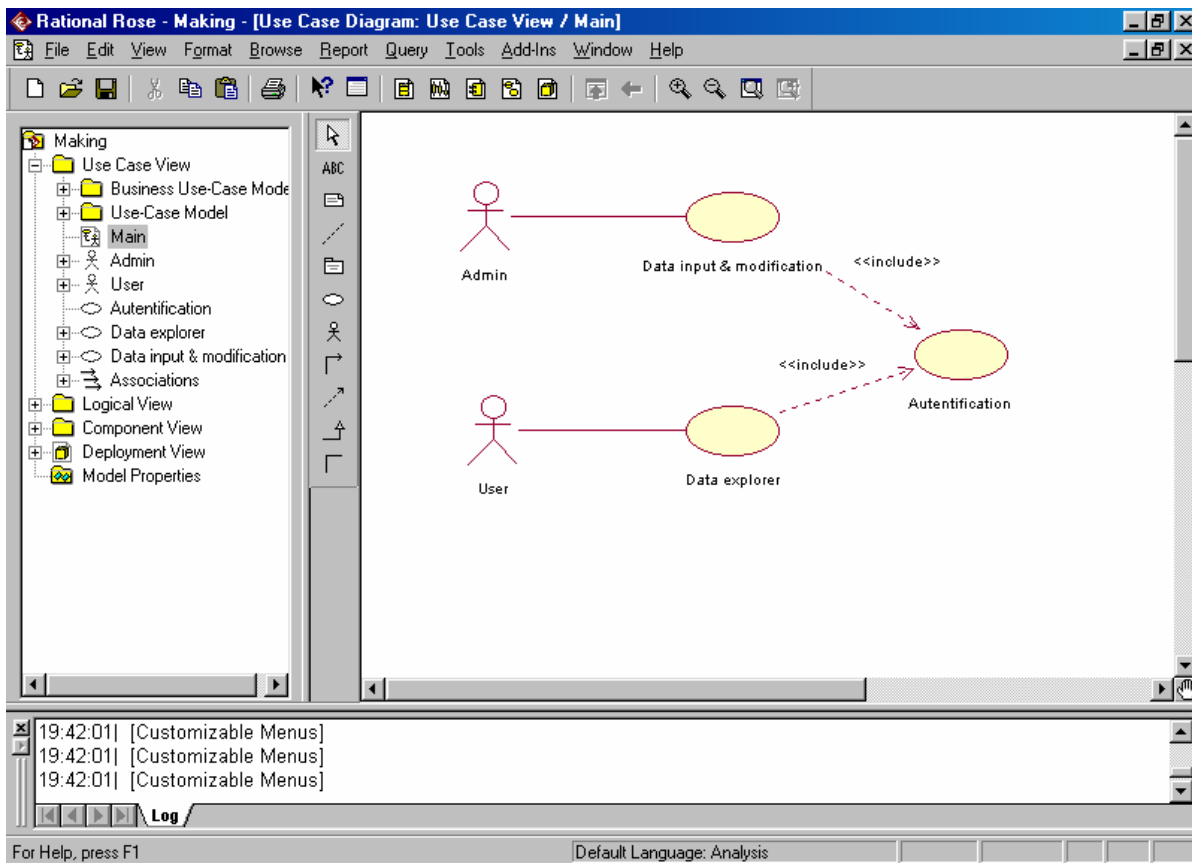


Рисунок 65 – Новый вариант использования

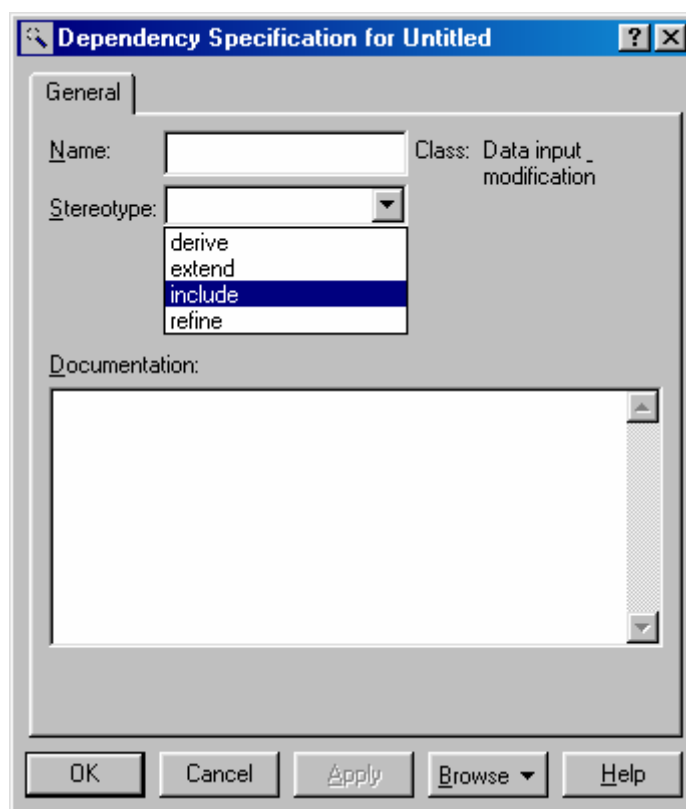


Рисунок 66 – Изменение стереотипа связи

Последнее изменение на диаграмме – добавление варианта использования «Формирование отчета» и связывание его с «Работой с данными» отношением зависимости типа «Расширение» (рис. 67).

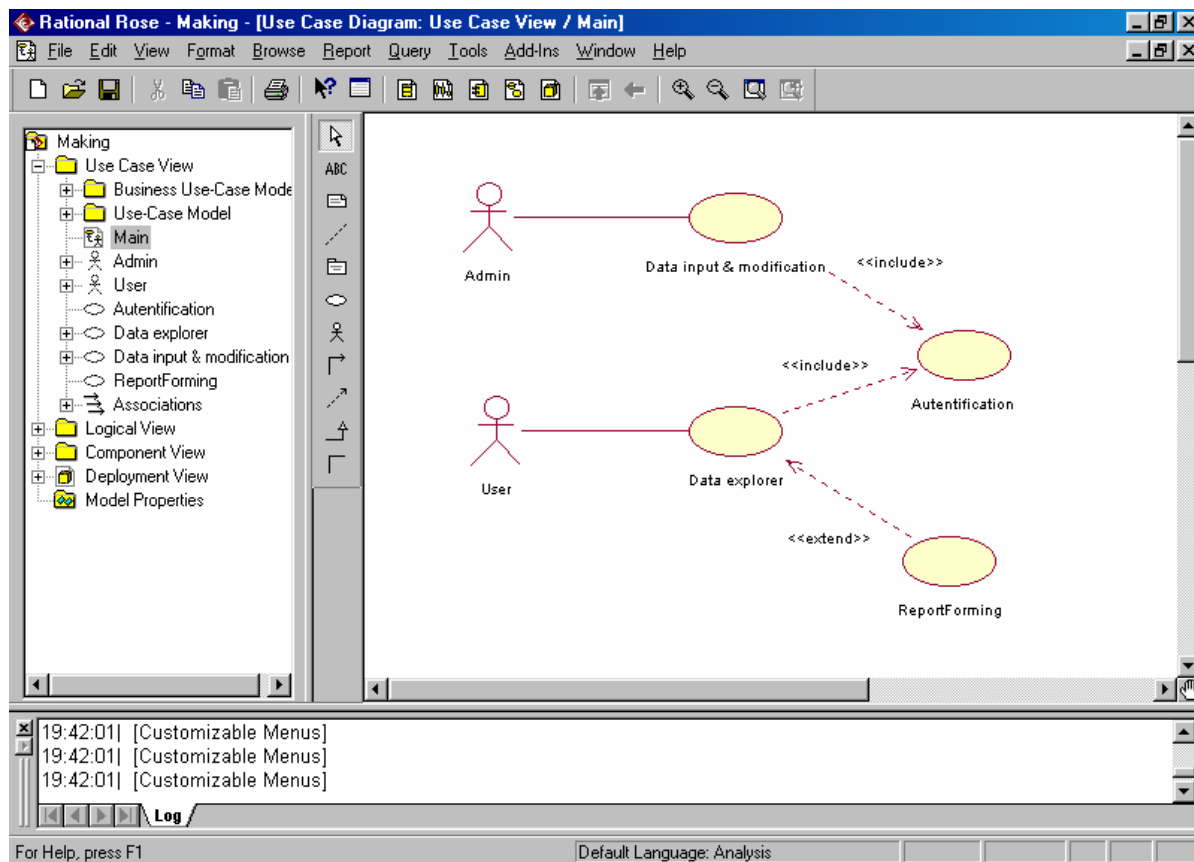


Рисунок 67 – Вариант использования «ReportForming»

При возникновении необходимости удаления элемента из модели простое выделение и нажатие клавиши «Delete» не приведет к желаемому результату – исчезнет только его изображение. Полное удаление элемента осуществляется в окне браузера проекта – контекстное меню, пункт «Delete».

Следующий этап – построение диаграммы классов. Выберем в главном меню пункт «Browse / Class Diagram» (или выберем слева «Logical View / Welcome») – на экране появится новая диаграмма. Присвоенное по умолчанию название «Welcome» лучше изменить при помощи контекстного меню (пункт «Rename») на более подходящее по смыслу (рис. 68). Построение диаграммы начинается с размещения нового класса.

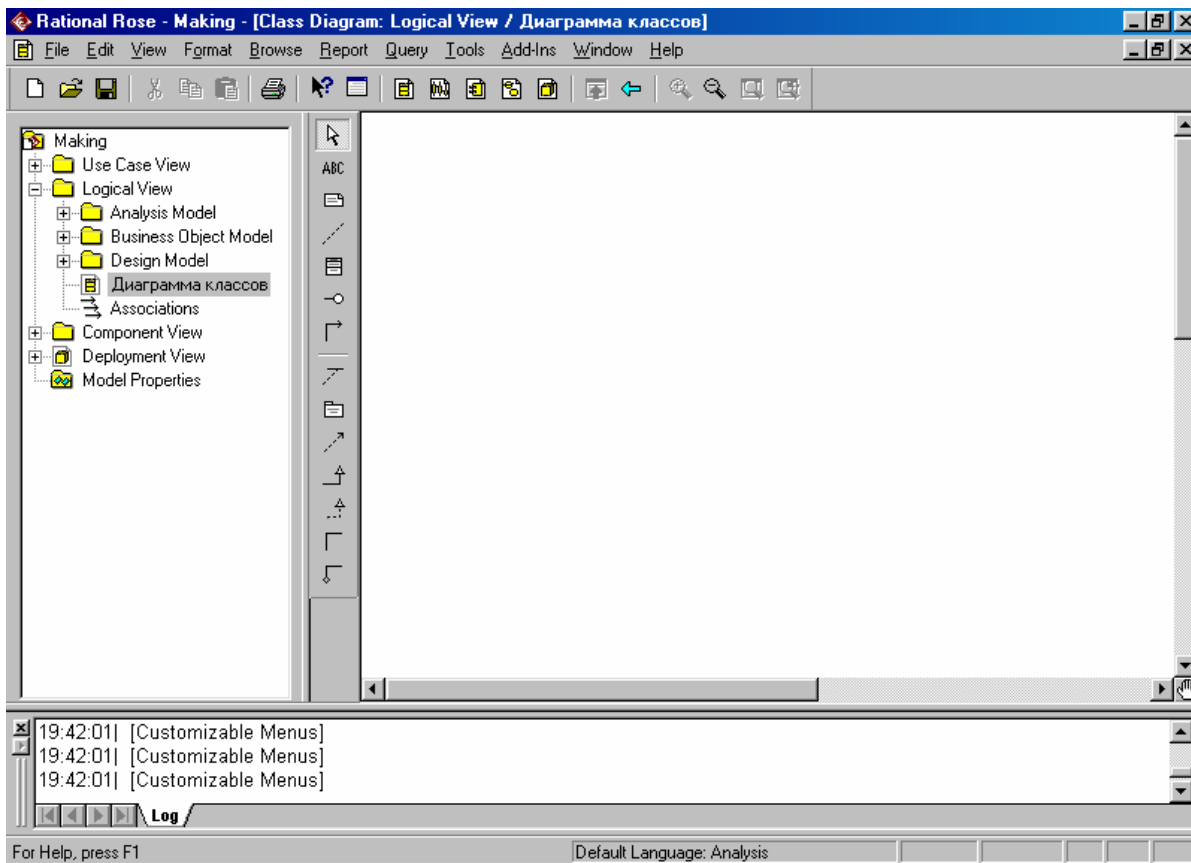


Рисунок 68 – Начало построения диаграммы классов

Нам будет предложен выбор: ввести имя нового класса или воспользоваться существующим (актеры из диаграммы вариантов использования автоматические предлагаются в качестве классов). Сначала введем класс `Admin`, основой которого является соответствующий актер (рис. 69).

После создания классов, описывающих обоих наших актеров, введем новый класс – программу. В окне спецификации класса напомним его имя (`Program`) и выберем стереотип (`control`), поскольку класс является управляющим (рис. 70). В результате диаграмма примет вид, показанный на рис. 71. Неудобство такого представления управляющего класса скажется при добавлении атрибутов и операций, поэтому посредством контекстного меню («Options / Stereotype Display / Label») придадим ему стандартный вид.

Добавление атрибутов и операций класса можно либо в окне спецификации класса («Attributes» и «Operations»), либо с помощью контекстного меню («New attribute» и «New operation»).

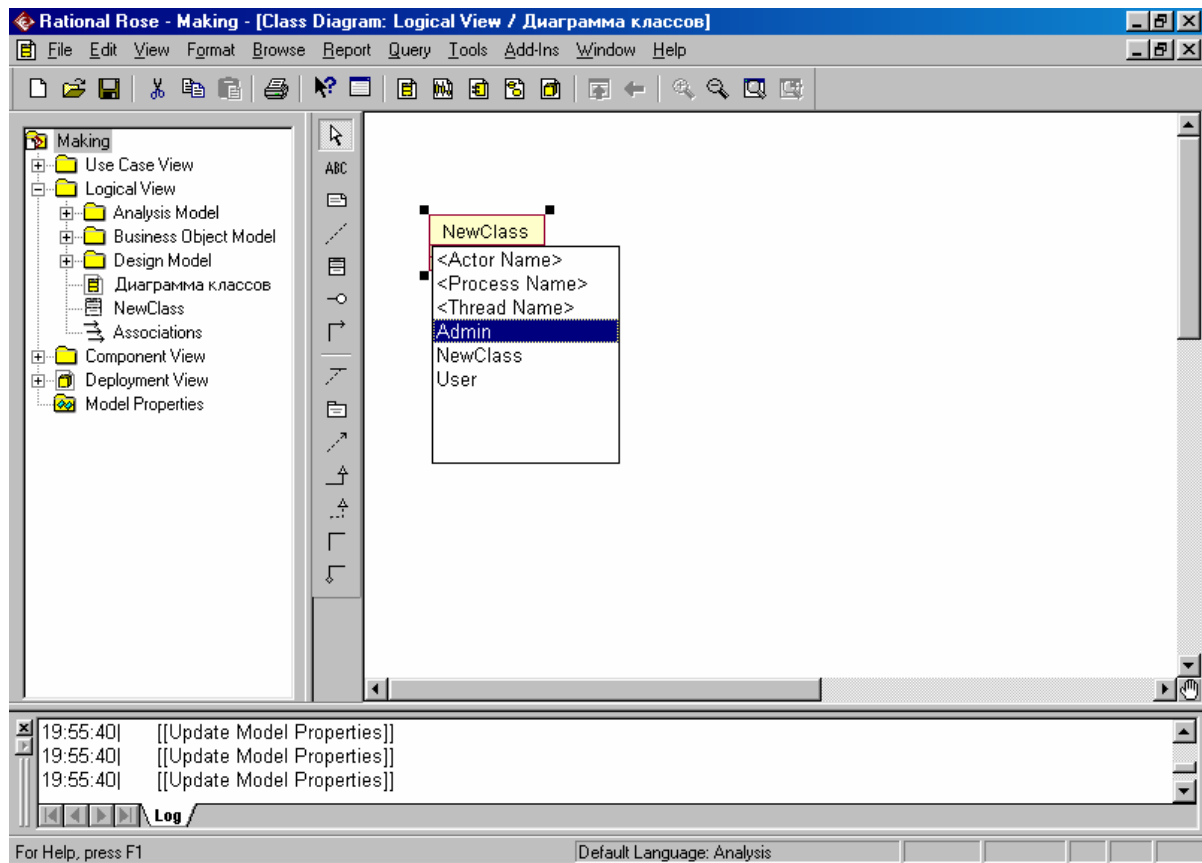


Рисунок 69 – Размещение нового класса - Admin

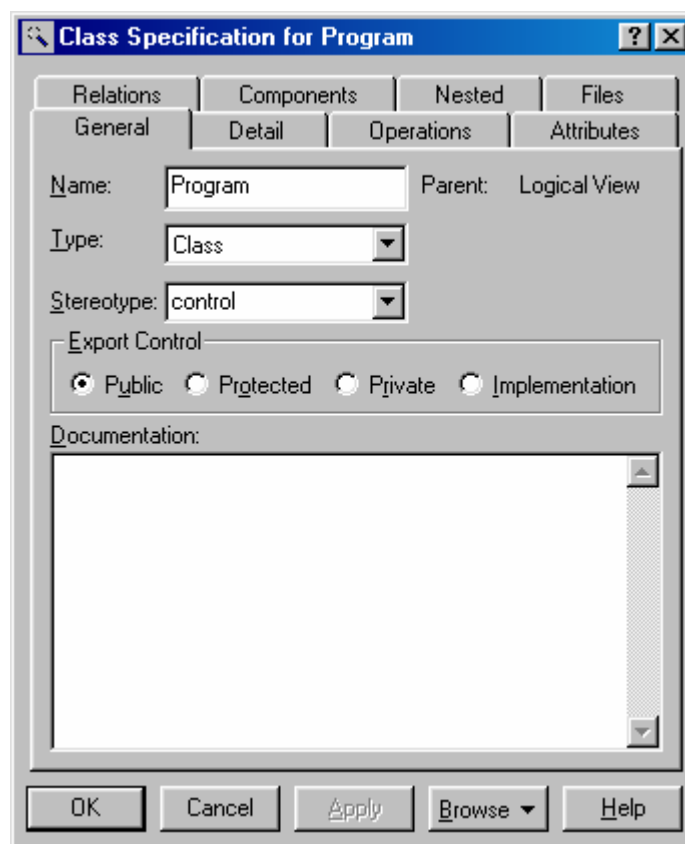


Рисунок 70 – Окно спецификаций класса

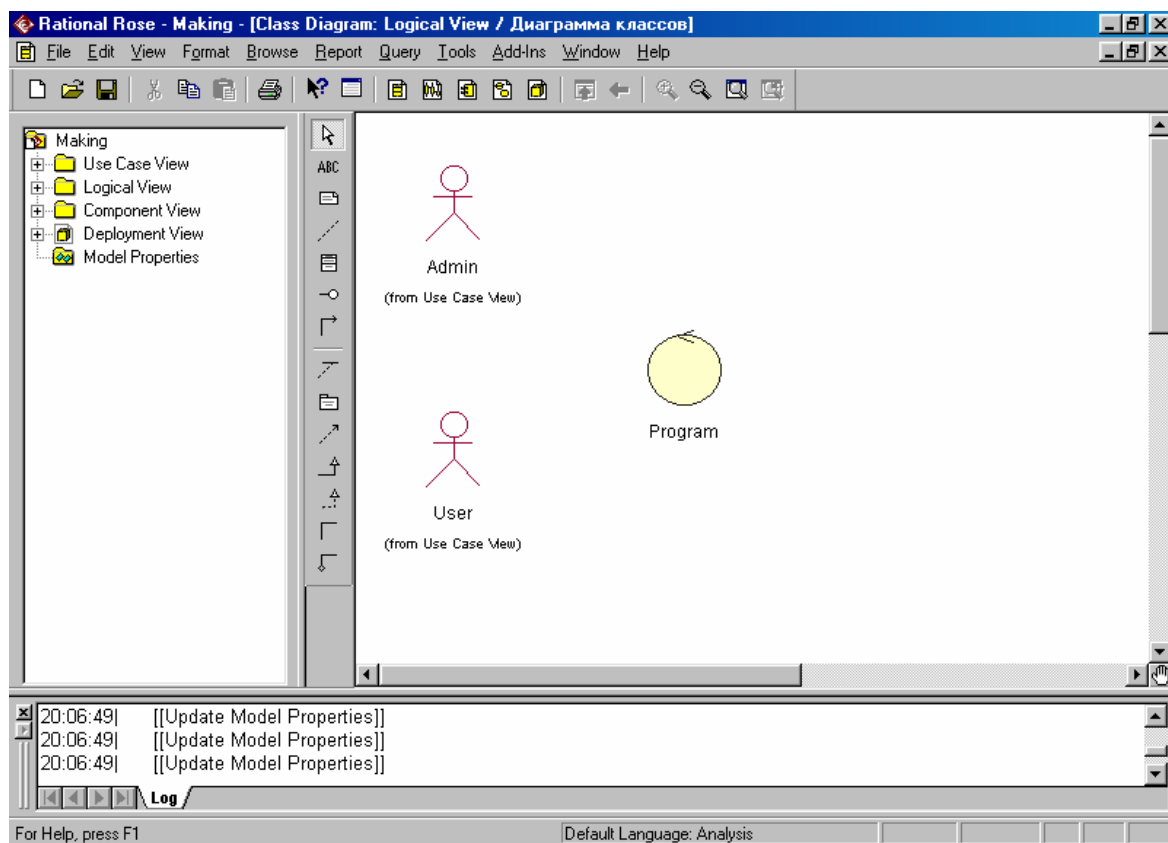


Рисунок 71 – Специальное изображение управляющего класса

Введение атрибутов главной программы в нашем примере нецелесообразно, а вот операцию «Авторизация пользователя» добавить необходимо. Тип возвращаемого значения укажем логическим (Boolean), видимость – «доступно для всех» (public). Результат действий приведен на рис. 72.

Теперь остается добавить два новых класса – «База данных» (атрибут – «Данные», операции – «ввестиДанные()», «изменитьДанные()» и «извлечьДанные()») и «Отчет» (атрибут – «Данные», операции – «сформировать()», «распечатать()» и «экспортировать()»), а также связи между ними (рис. 73).

После окончания работы с диаграммой классов приступаем к построению диаграмм взаимодействия, а именно – к диаграмме кооперации. Рассмотрим только один пример работы системы – работу пользователя. Сначала тот проходит авторизацию в программе, затем должен извлечь информацию из базы данных, сформировать и распечатать отчет.

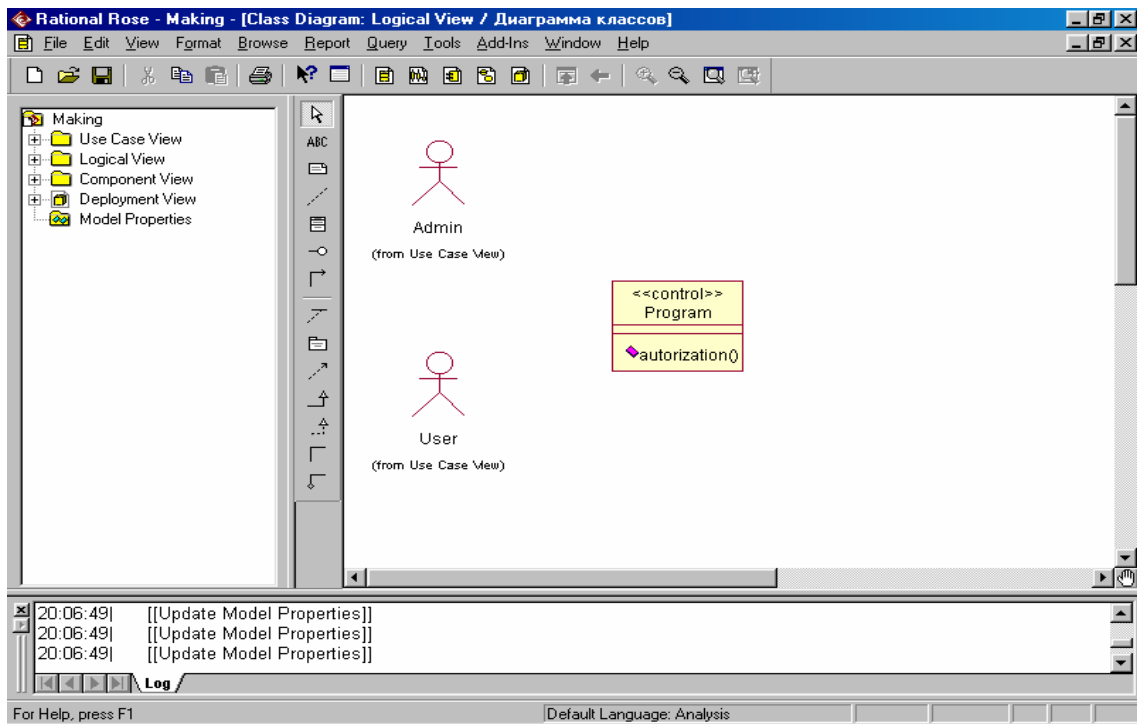


Рисунок 72 – Стандартное изображение управляющего класса с операцией

Создание диаграммы происходит в следующей последовательности: пункт главного меню пункт «Browse / Interaction Diagram», «New / Ok», ввод имени («Диаграмма кооперации») и выбор типа диаграммы («Diagram type: Collaboration») – рис. 74.

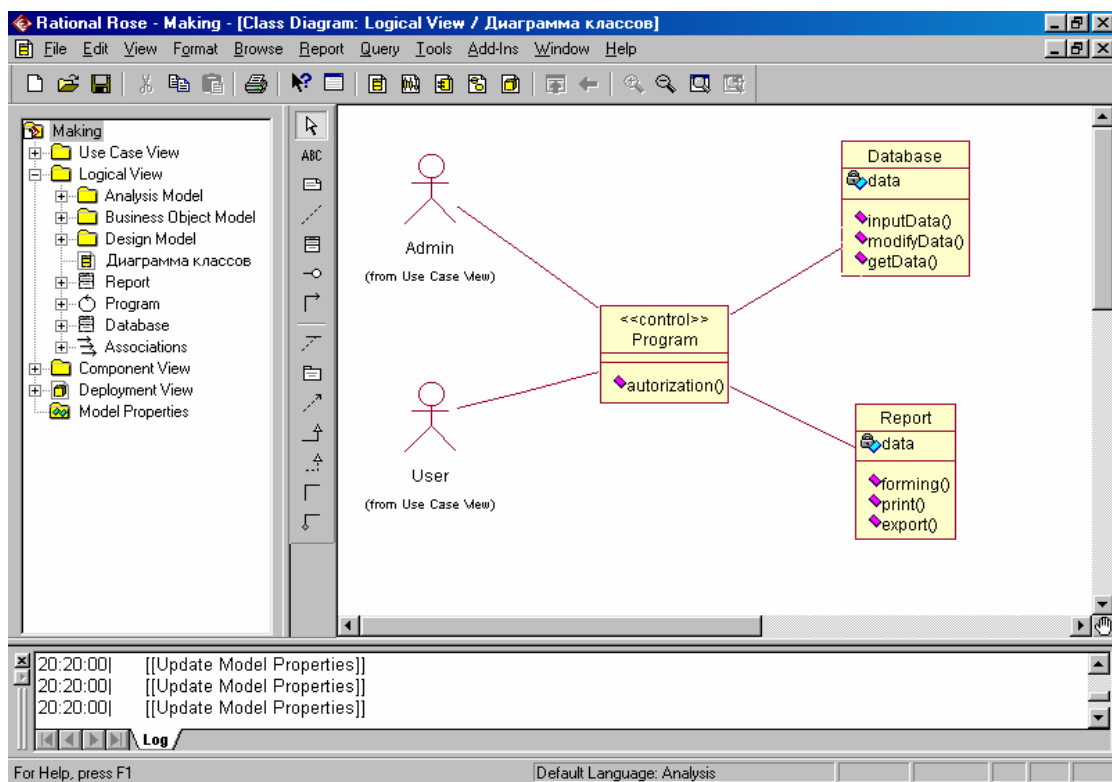


Рисунок 73 – Окончательный вид диаграммы классов

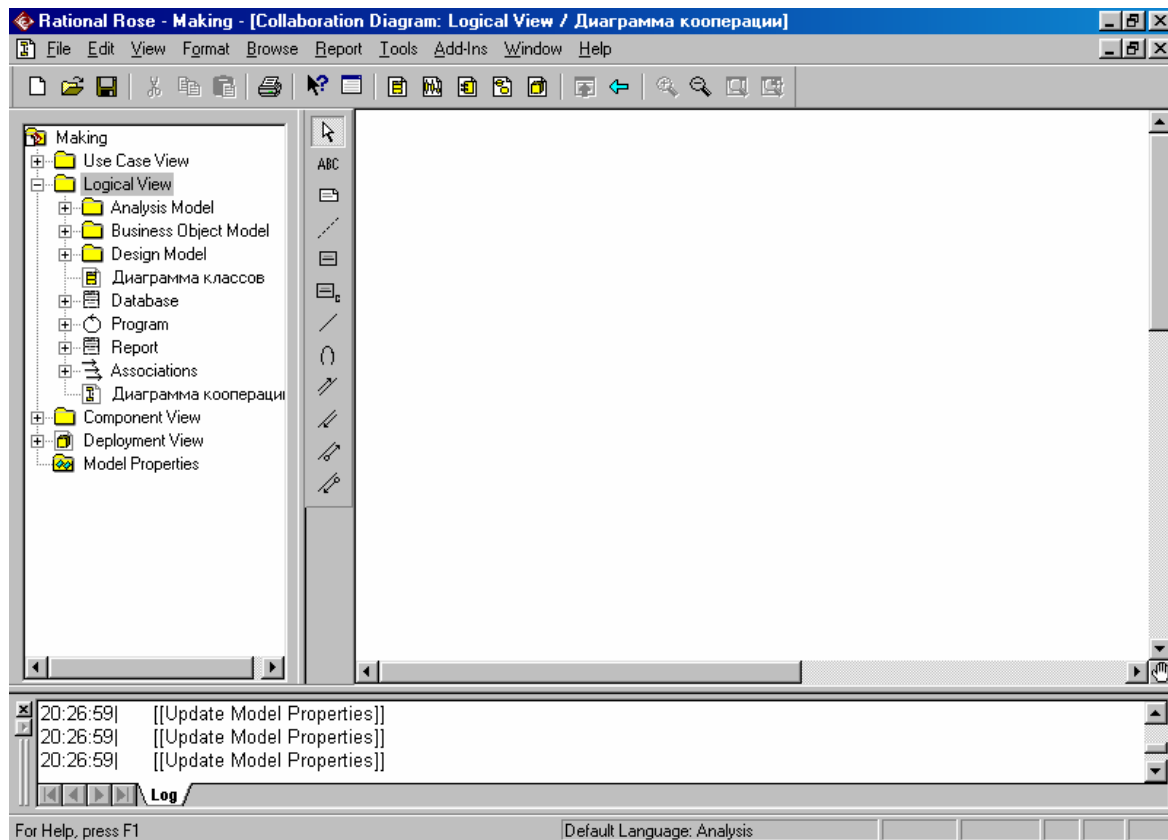


Рисунок 74 – Начальный вид диаграммы кооперации

Поместим в окно диаграммы новый объект и выберем в его окне спецификации интересующий нас тип – User (рис. 75). Поскольку собственное имя пользователя нам в данном случае совершенно неважно, строчку «Name» оставим пустой (анонимный объект). Аналогичное действие осуществим по отношению к объекту класса «Программа» (менять внешний вид объекта здесь нет необходимости) и соединим два объекта линией («Object Link») – рис. 76.

Поместить на линию связи сообщение можно двумя способами. Первый – вызвав двойным щелчком мыши на линии окно ее спецификации, в разделе «Messages» при помощи контекстного меню добавить сообщение (выбрать при этом пункт «Insert To: Program», то есть явно указать направление). При этом доступные операции будут показаны в виде выпадающего меню (рис. 77).

Второй способ – выделить на специальной панели инструментов Link message и поместить его на линию связи. Затем в окне спецификации этого сообщения выбрать из выпадающего меню доступных операций ту, которая интересует нас. Результат будет идентичен первому (рис. 78).

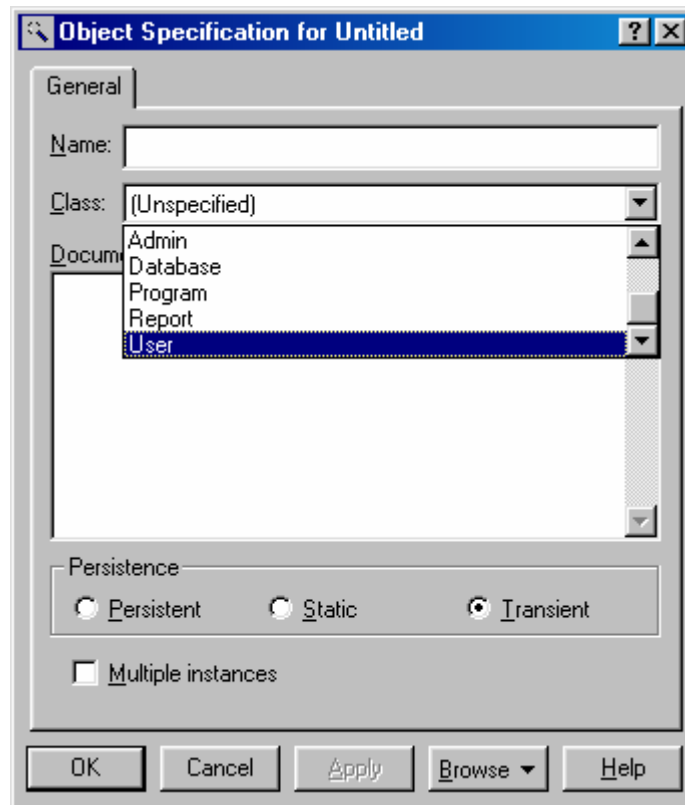


Рисунок 75 – Окно спецификации объекта

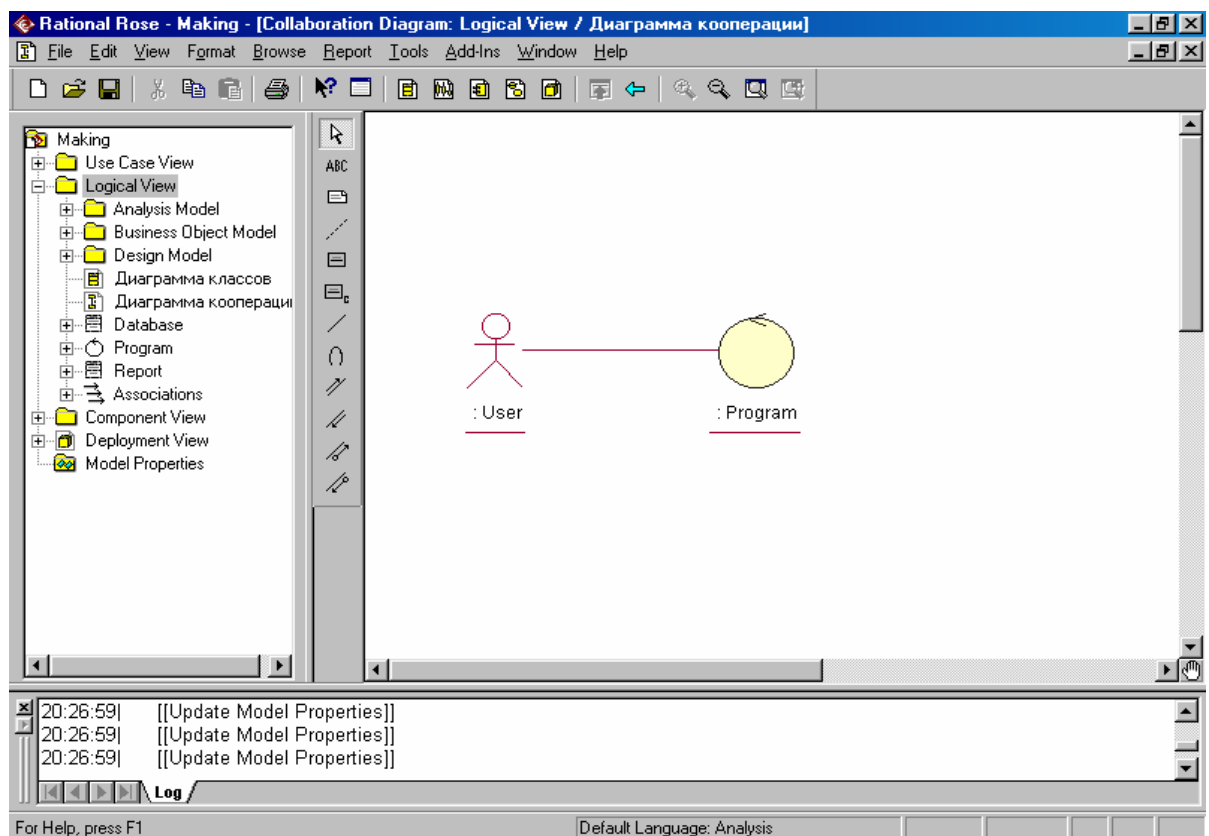


Рисунок 76 – Два объекта на диаграмме кооперации

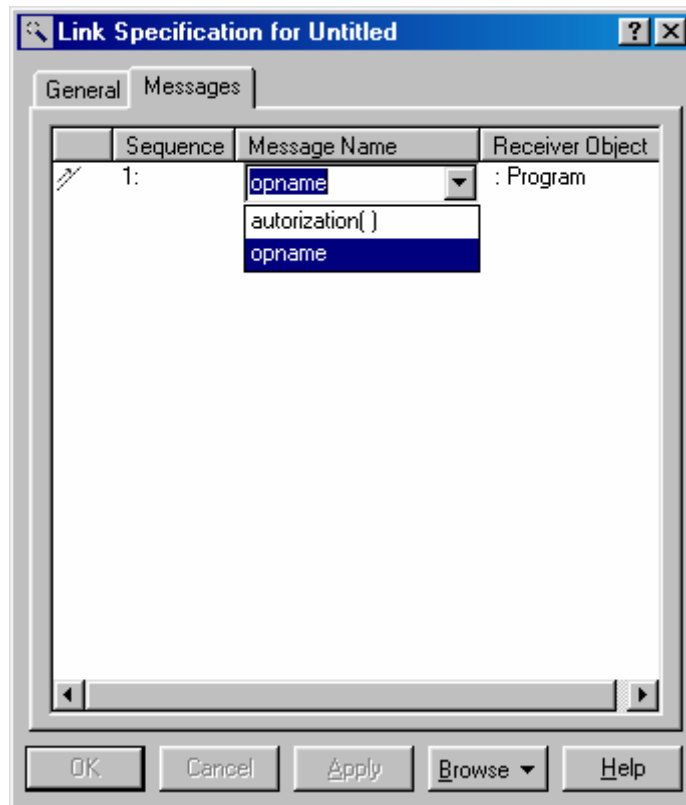


Рисунок 77 – Добавление сообщения в окне спецификации связи

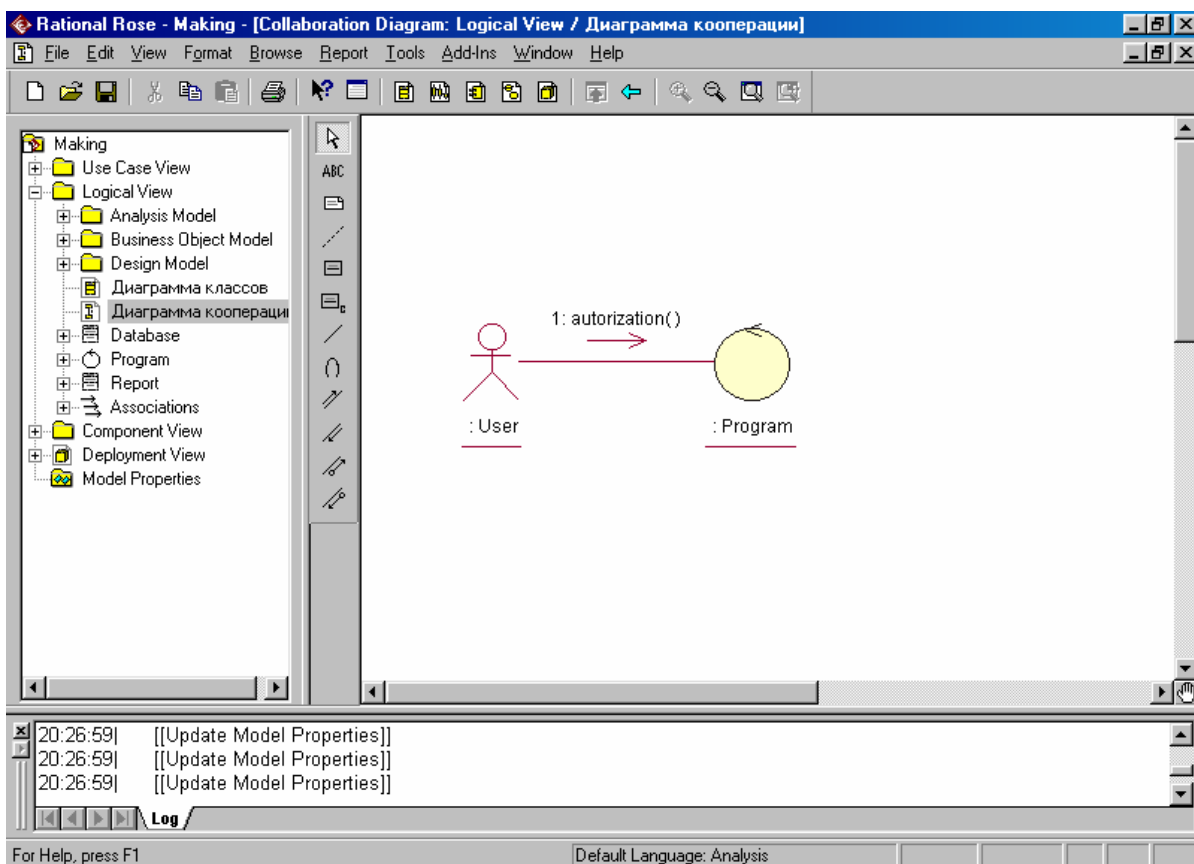


Рисунок 78 – Сообщение на диаграмме кооперации

Теперь поместим на диаграмму два оставшихся объекта (анонимные объекты классов «База Данных» и «Отчет») и свяжем их с программой (рис. 79).

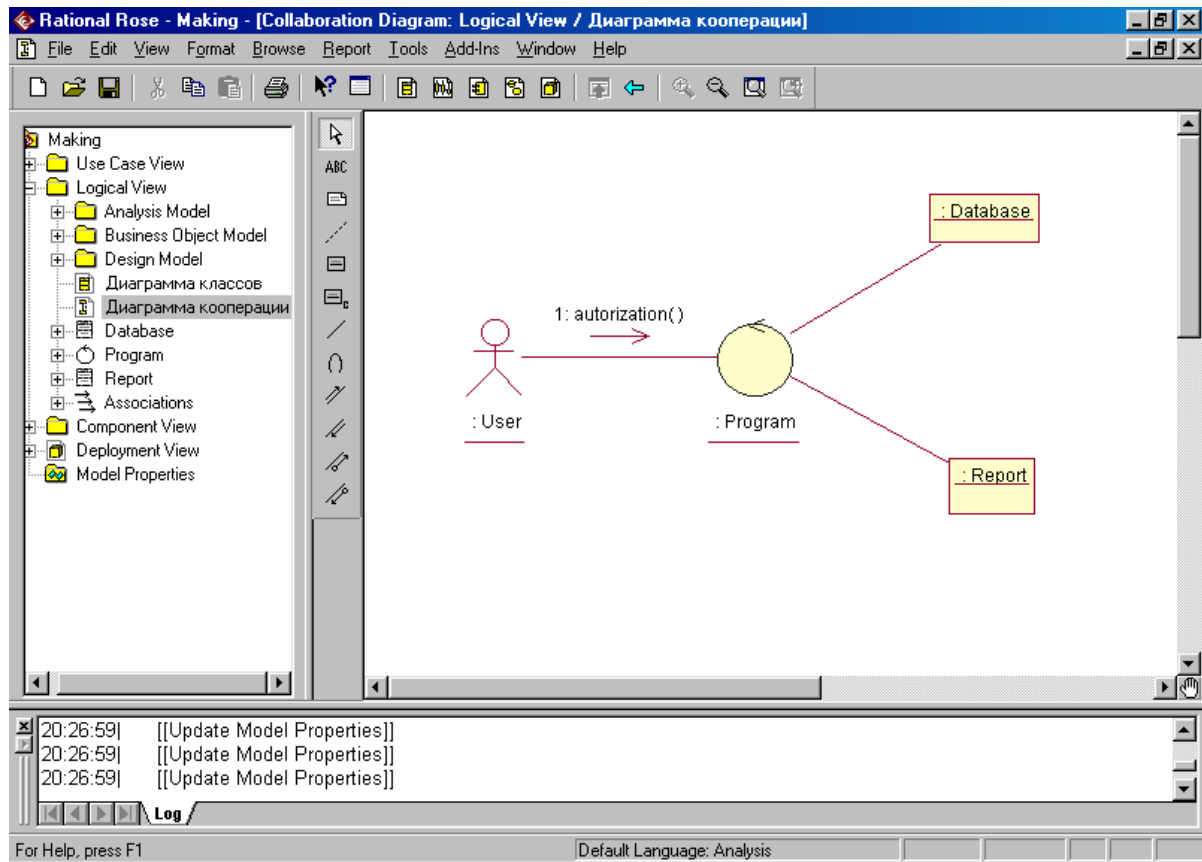


Рисунок 79 – Все размещенные объекты на диаграмме кооперации

Вторым сообщением работающей системы будет, очевидно, запрос от программы к «Базе данных» – вызов операции «извлечьДанные()», третьим – запрос от программы к «Отчету» – вызов операции «сформировать()», четвертым – запрос к тому же объекту вызова операции «печать()». Окончательный вид диаграммы кооперации представлен на рис. 80.

Диаграмма последовательности формируется на основании диаграммы кооперации пунктом меню «Browse / Create Sequence Diagram» или просто нажатием клавиши F5. Как видно на рис. 81, в окне браузера проектов теперь расположились две «Диаграммы кооперации», что, по сути, неверно; присвоить диаграмме новое имя можно при помощи контекстного меню (пункт «Rename»).

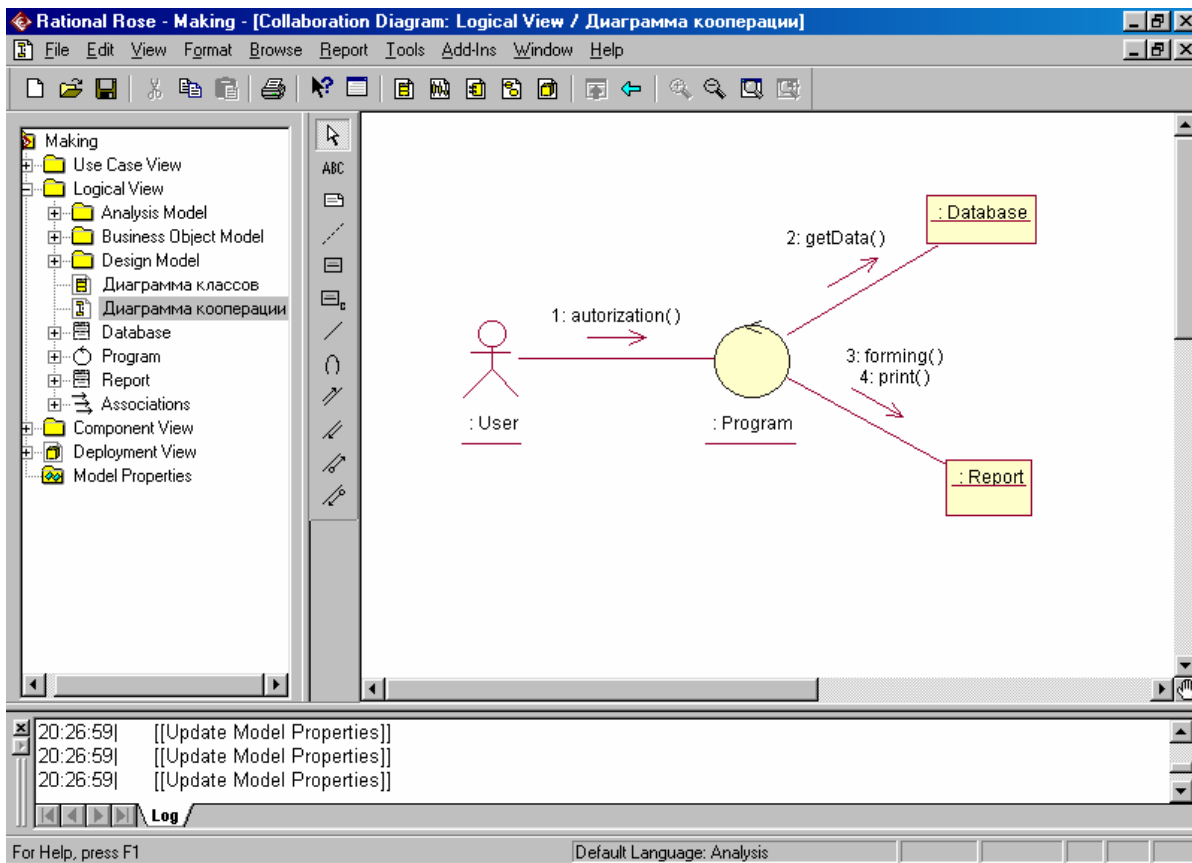


Рисунок 80 – Окончательный вид диаграммы кооперации

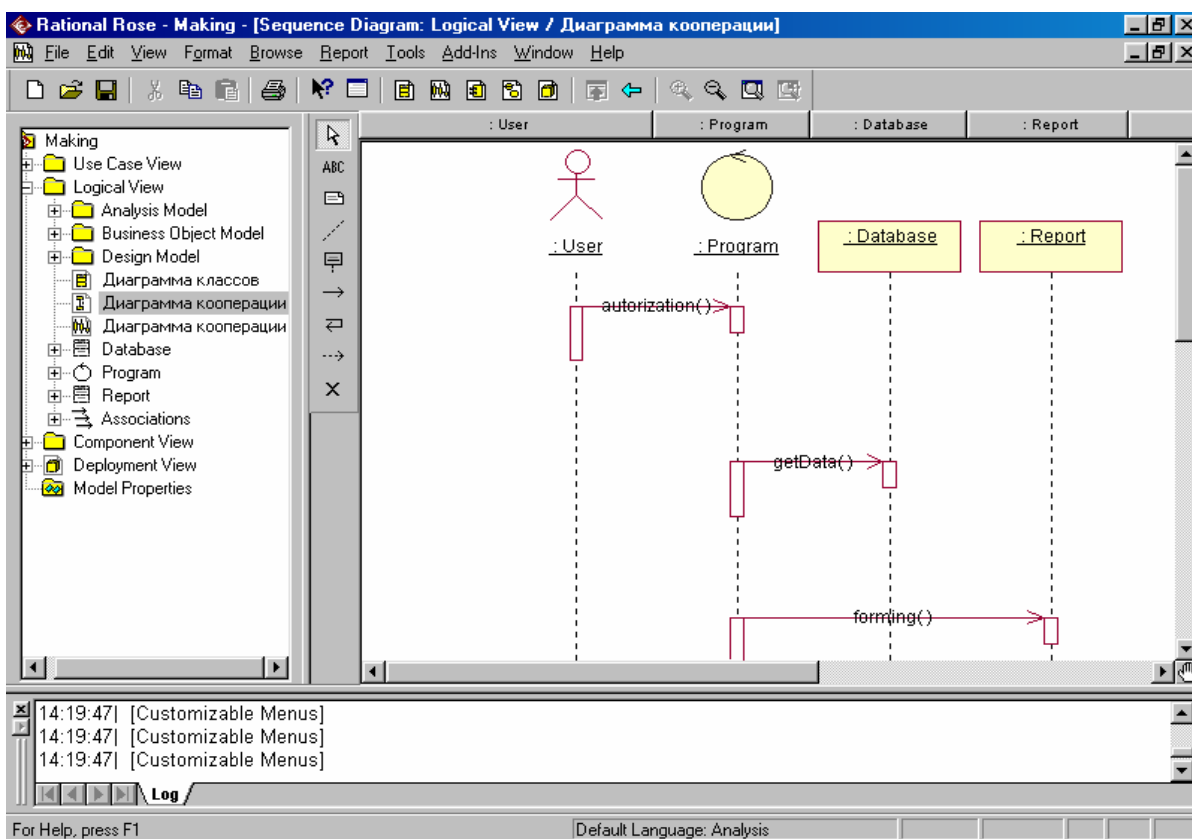


Рисунок 81 – Автоматически сформированная диаграмма последовательности

Необходимо выполнить еще несколько корректировок диаграммы последовательности, большая часть которых носит «косметический» характер. Окончательный вид см. на рис. 82.

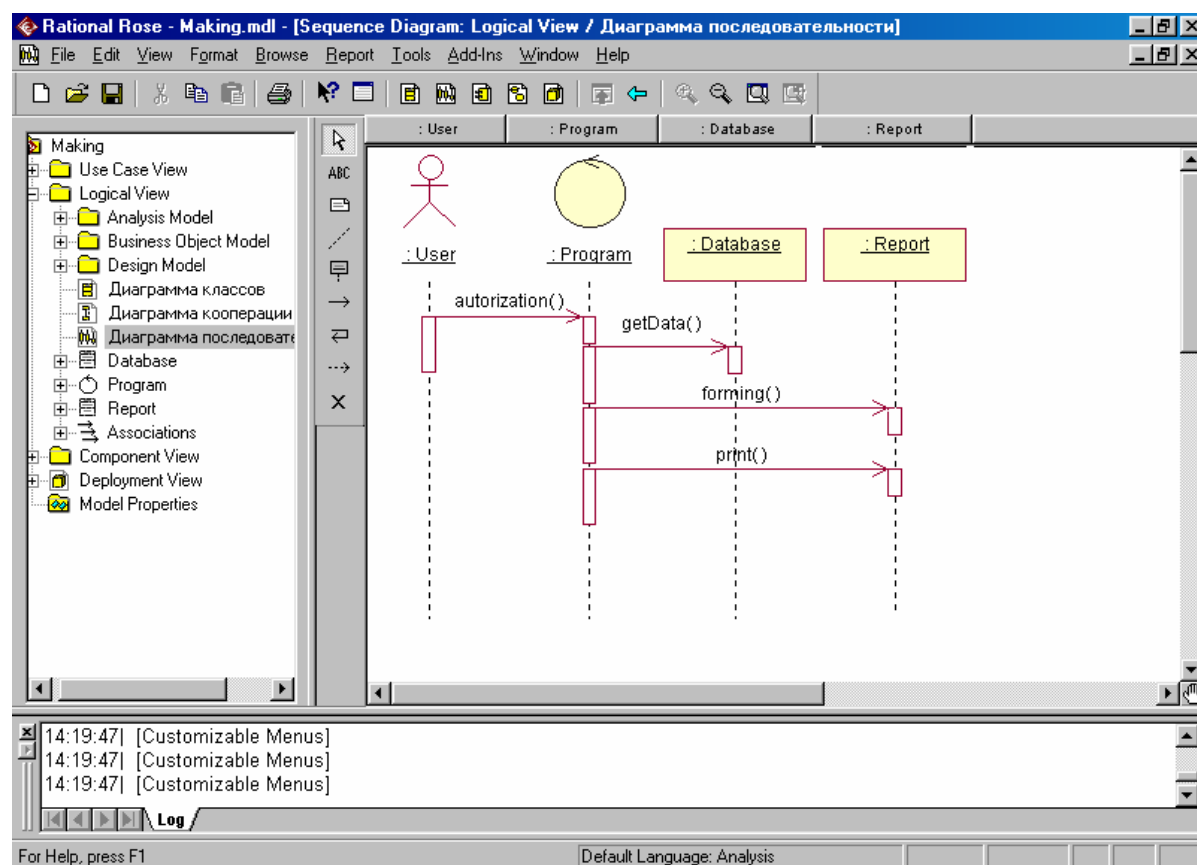


Рисунок 82 – Окончательный вид диаграммы последовательности

Диаграмма состояний создается пунктом меню «Browse / State Machine Diagram» (или нажатием клавиш Ctrl+T), «New / Ok», затем нужно ввести имя диаграммы («Диаграмма состояний») и ее тип («Diagram Type: Statechart»). В браузере проектов новая диаграмма разместится в ветви «Logical View / State-Activity Model» (рис. 83).

Очевидно, что наша система может находиться в семи различных состояниях (не считая начального и конечного): «Ожидание ввода пароля», «Проверка пароля», «Выбор данных для отчета», «Формирование отчета», «Ожидание выбора» (куда направлять отчет), «Печать отчета» и «Экспорт отчета».

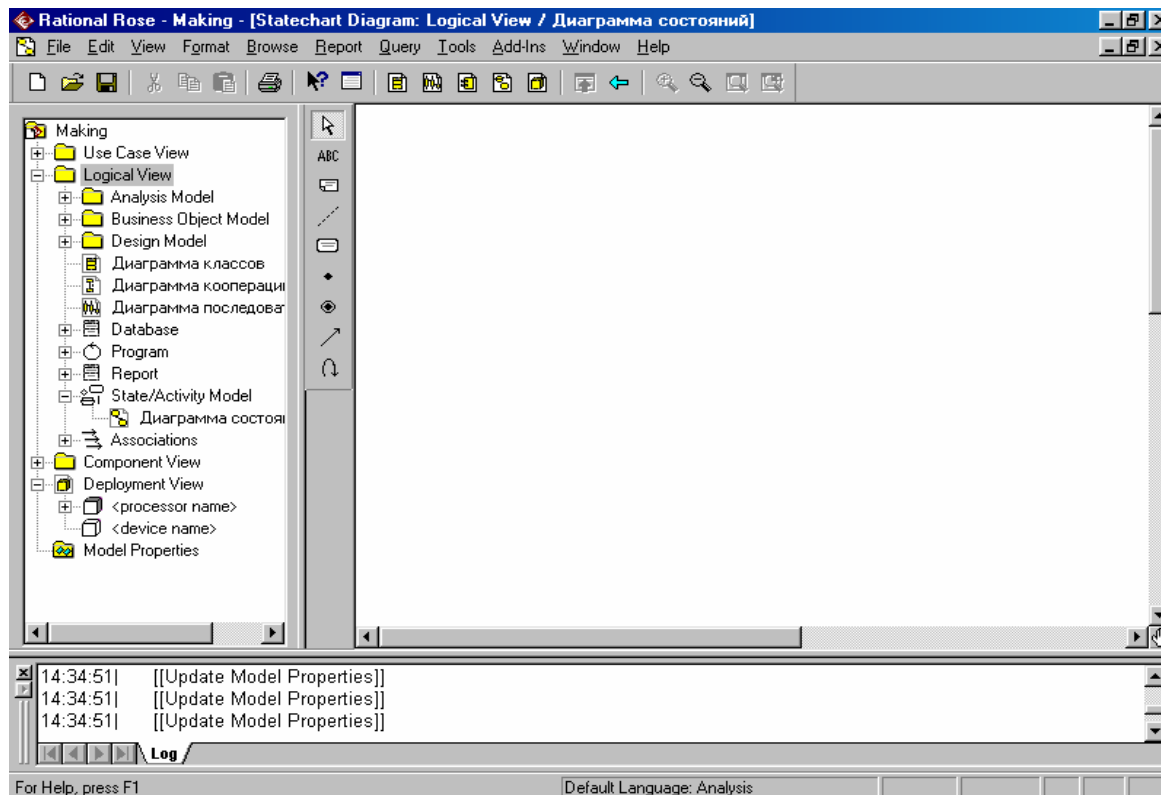


Рисунок 83 – Начальный вид диаграммы состояний

Поместим на диаграмму начальное (черный кружок) и первое состояния (рис. 84); очевидно, что его следует назвать «Ожидание ввода пароля» (имя вводится двойным щелчком мыши или посредством окна спецификации состояния).

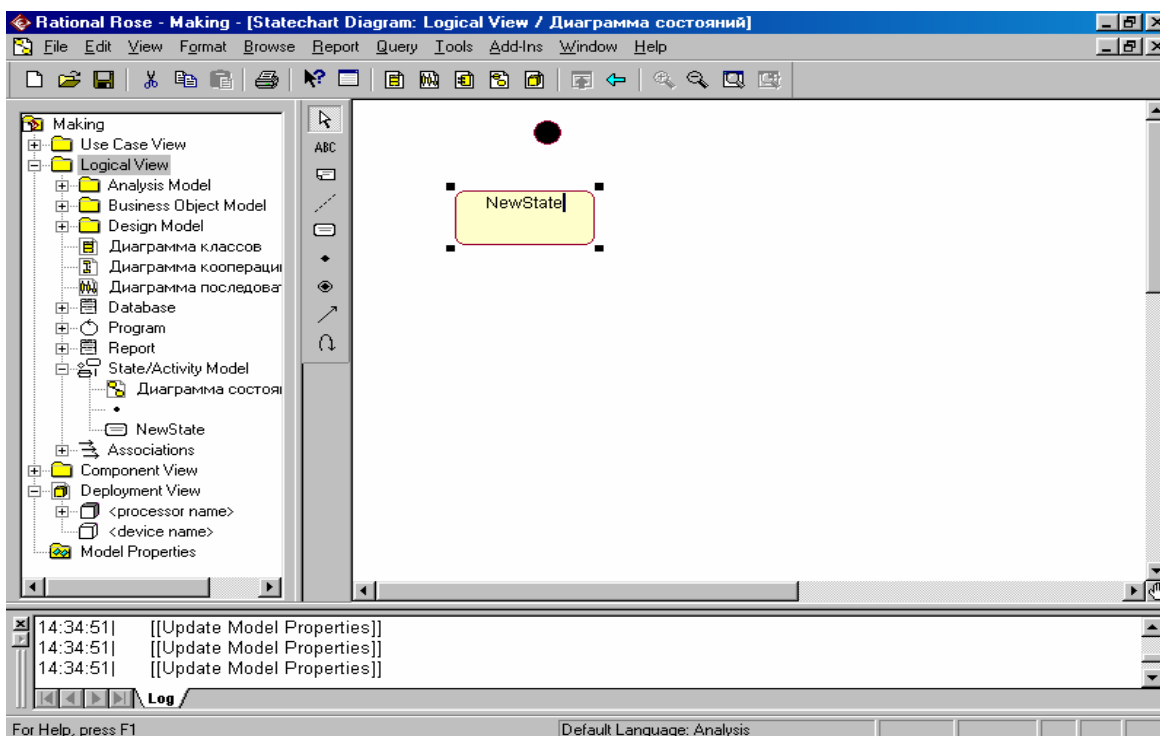


Рисунок 84 – Добавление нового состояния

Соединим начальное и первое состояние линией связи (State Transition), в окне спецификации перехода укажем название события (General / Event) – «Загрузка программы» (рис. 85).

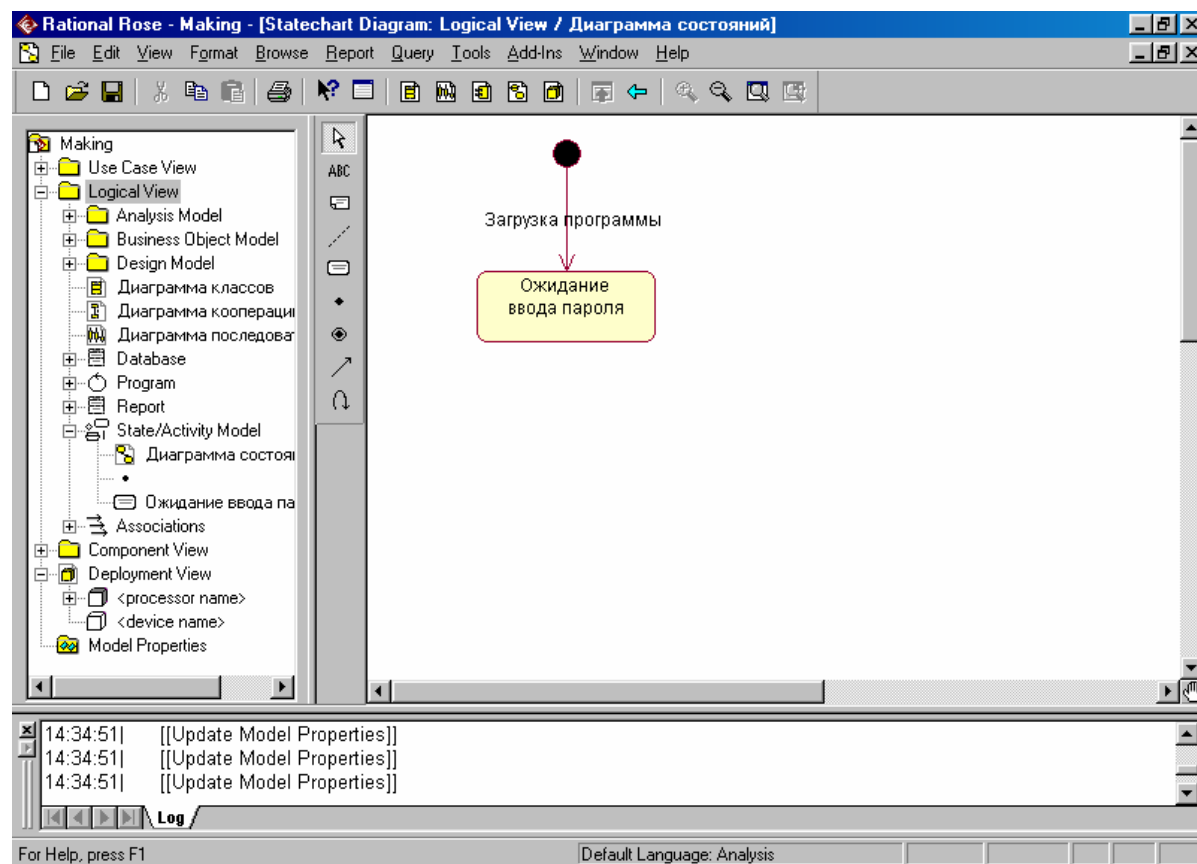


Рисунок 85 – Добавление первого перехода и первого события

Переход в новое состояние – «Проверка пароля» – может осуществиться при наступлении события «Пароль введен» (рис. 86).

Отсюда возможны сразу три перехода: если введен правильный пароль – на «Выбор данных для отчета», неправильный пароль – возврат в предыдущее состояние, неправильный пароль вводится подряд три раза – выход. Собственно событием здесь могут считаться «Три неправильных попытки», остальные условия введем как сторожевые в окне спецификации перехода (Detail / Guard Condition) (рис. 87).

Переход, вызванный тремя неправильными попытками, можно конкретизировать в окне его спецификации: кроме непосредственного названия, в разделе «Detail / Action» можно явно задать действие («Выход») (рис. 88).

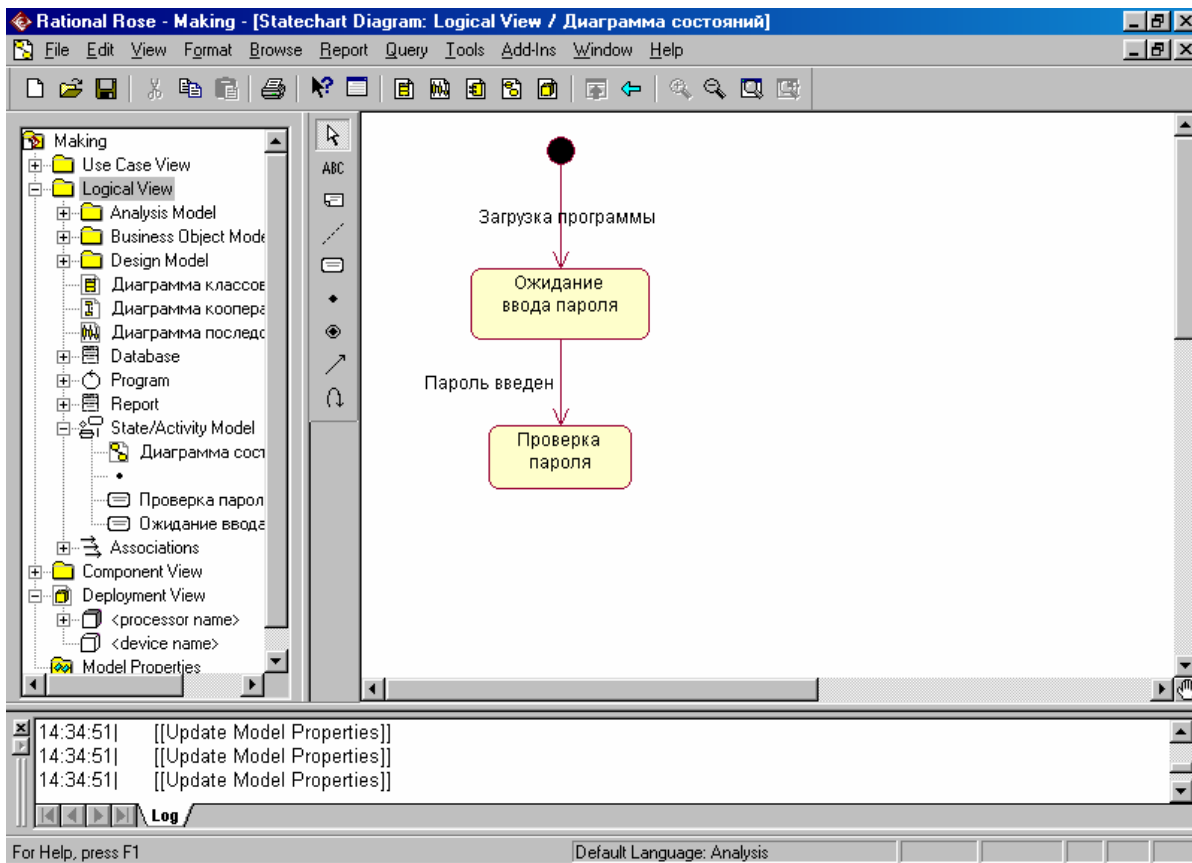


Рисунок 86 – Добавление нового состояния и перехода

The screenshot shows the "State Transition Specification" dialog box. The "General" tab is selected. The "Guard Condition" field contains the text "Пароль правильный" (Password is correct). The "Action" field is empty. The "Send event" field is empty. The "Send arguments" field is empty. The "Send target" field is empty. The "Transition between substates" section has two dropdown menus: "From" is set to "Проверка пароля" (Password check) and "To" is set to "Выбор данных для отчета" (Select data for report). The dialog has buttons for "OK", "Cancel", "Apply", "Browse", and "Help".

Рисунок 87 – Добавление сторожевого условия в окне спецификации перехода

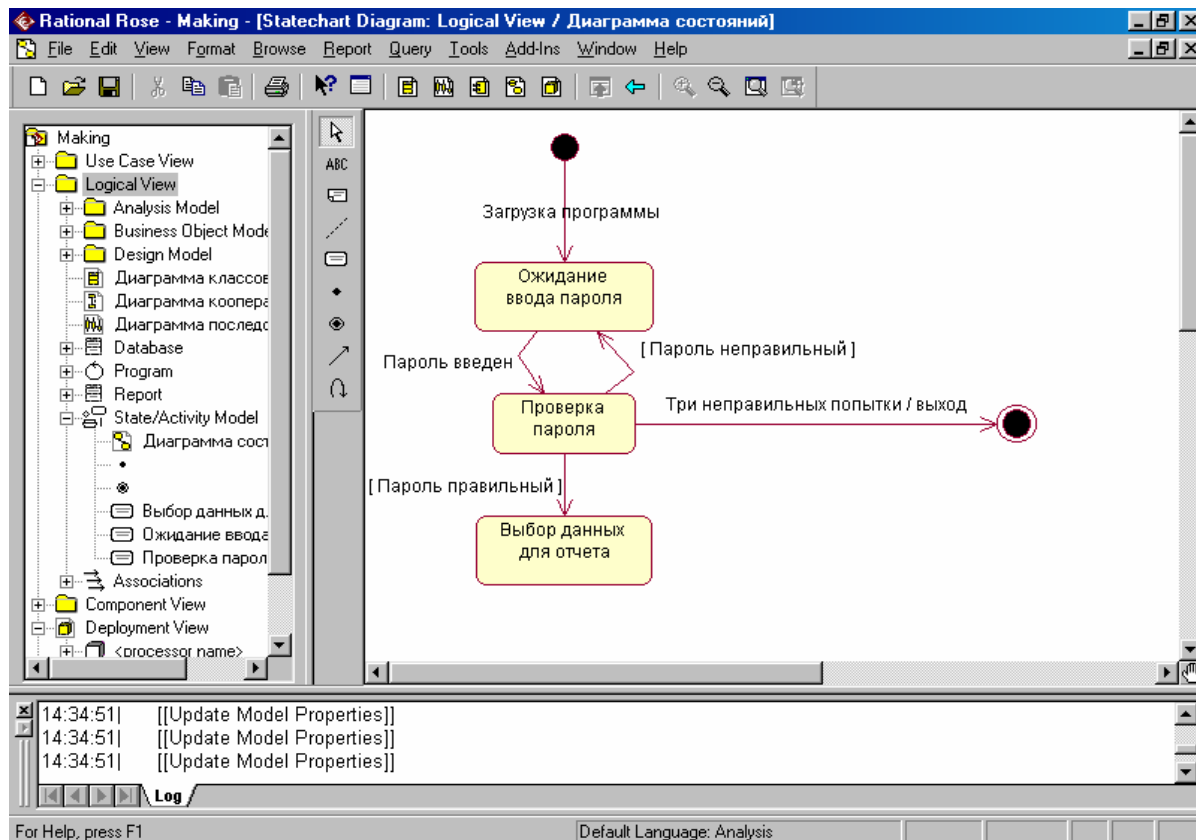


Рисунок 88 – Добавление новых состояний и переходов

Осталось добавить оставшиеся четыре состояния и переходы между ними (рис. 89).

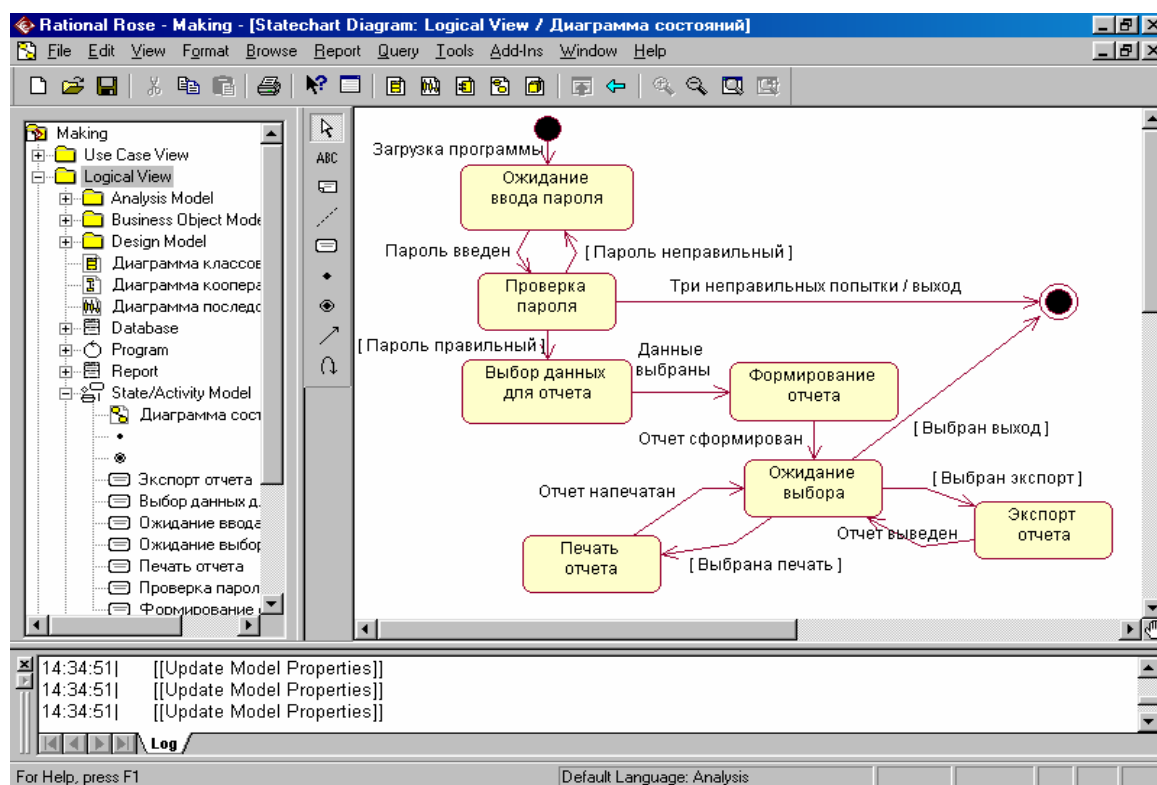


Рисунок 89 – Окончательный вид диаграммы состояний

Диаграмма деятельности создается тем же пунктом меню «Browse / State Machine Diagram», что и диаграмма состояния, но выбирается другой тип («Diagram Type: Statechart»). В браузере проектов новая диаграмма также разместится в ветви «Logical View / State-Activity Model» (рис. 90).

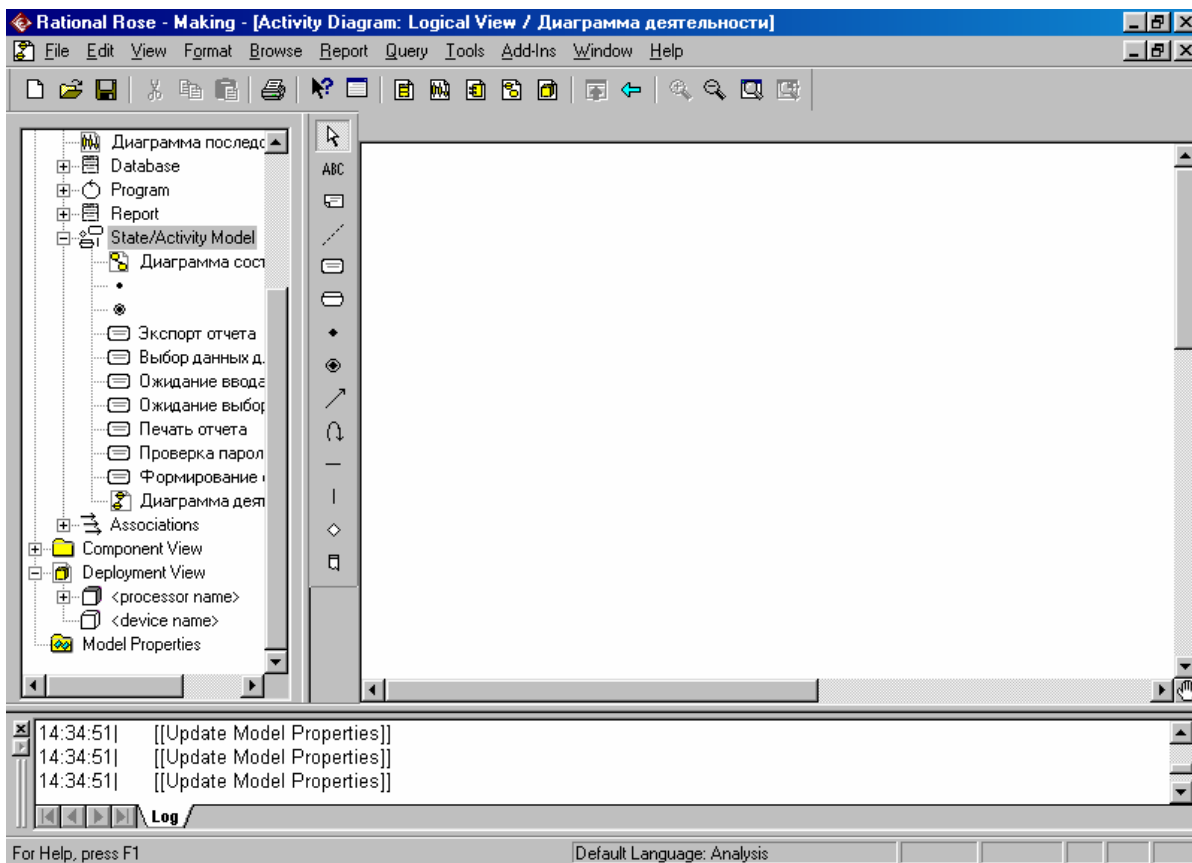


Рисунок 90 – Начальный вид диаграммы деятельности

Начальное состояние в модели может быть только одно, поэтому попытка взять его со специальной панели инструментов ни к чему не приведет: начальное состояние нужно найти в окне браузера проекта (черный кружок сразу после названия диаграммы состояний) и «перетянуть» в окно диаграммы.

Логика построения диаграммы деятельности практически полностью повторяет логику построения диаграммы состояний (строго говоря, в данном примере эта диаграмму можно было и опустить). Первое действие – ввод пароля – выбираем на специальной панели инструментов (пункт Activity), размещаем в окне диаграммы и присваиваем имя (рис. 91).

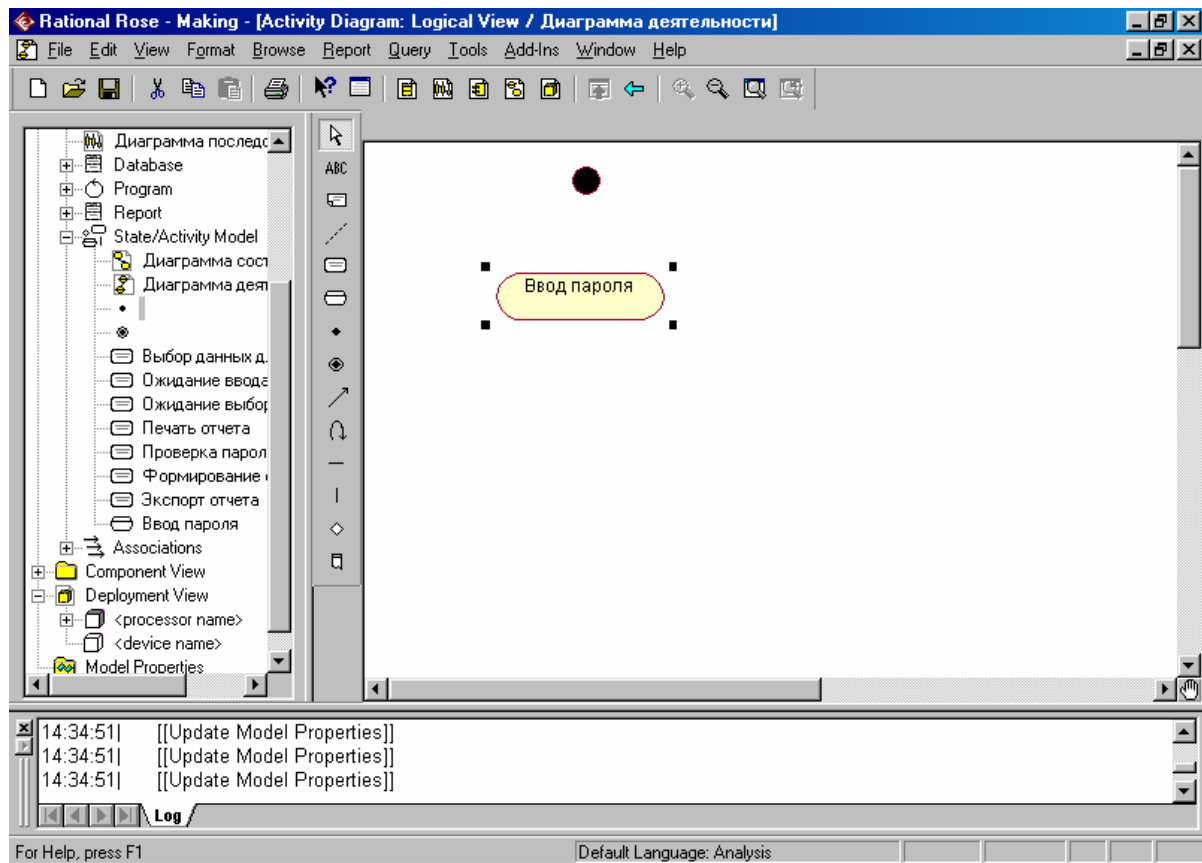


Рисунок 91 – Добавление нового действия

Соединяем начальное состояние с первым действием; имени события на переходе указывать не нужно. Далее на диаграмме начинается ветвление: добавляется знак «Decision», из которого возможны два перехода со сторожевыми условиями «Пароль правильный» и «Пароль неправильный». Во втором случае сразу можно добавить переход к действию «Выход из программы» и окончанию работы (рис. 92).

В случае правильности пароля добавим действия «Извлечение данных для отчета», «Создание отчета» и «Вывод отчета» (ветвление для разделения случаев «Печать отчета» и «Экспорт отчета» опустим). Окончательный вид диаграммы деятельности представлен на рис. 93.

Диаграмма компонентов создается пунктом меню «Browse / Component Diagram», в правом окне появится заготовка новой диаграммы (рис. 94). Расположенный там по умолчанию пакет «Implementation model» можно удалить (окно браузера проекта, контекстное меню, пункт «Delete»).

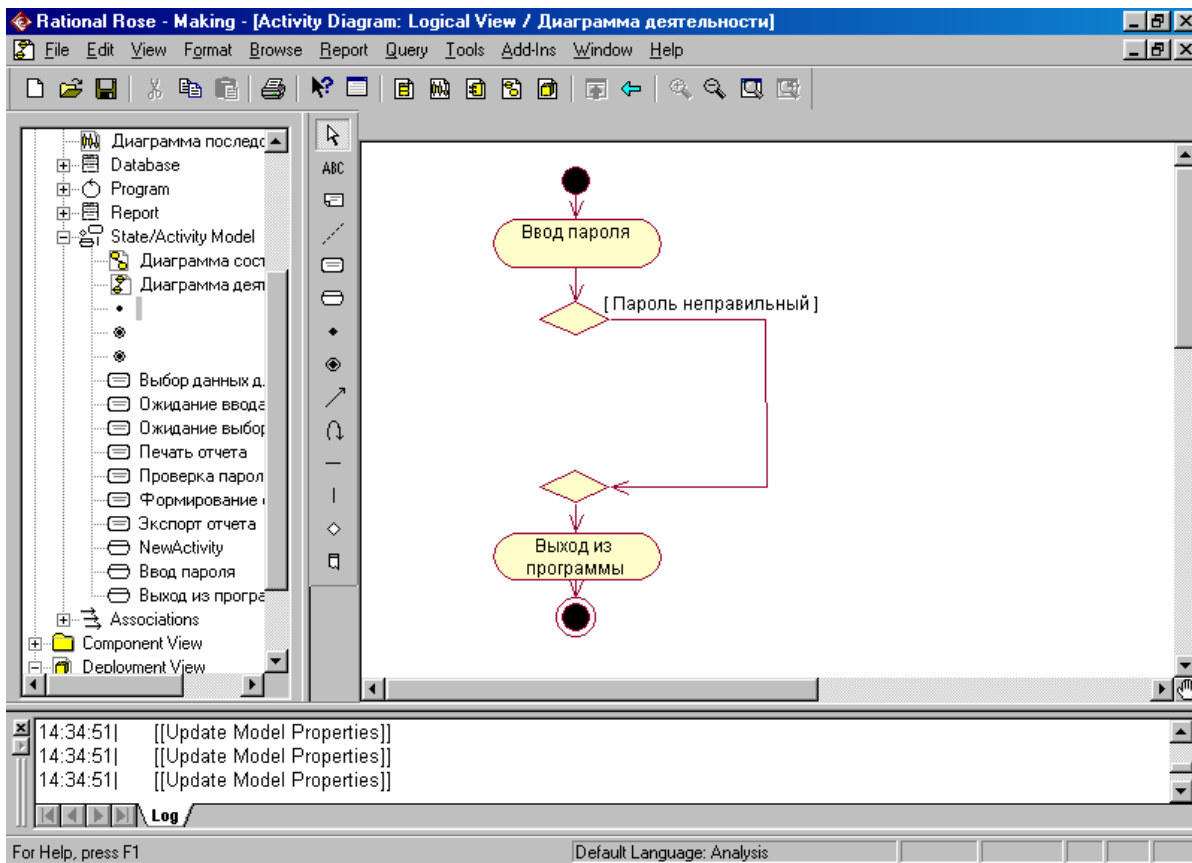


Рисунок 92 – Добавление ветвления и последних действий

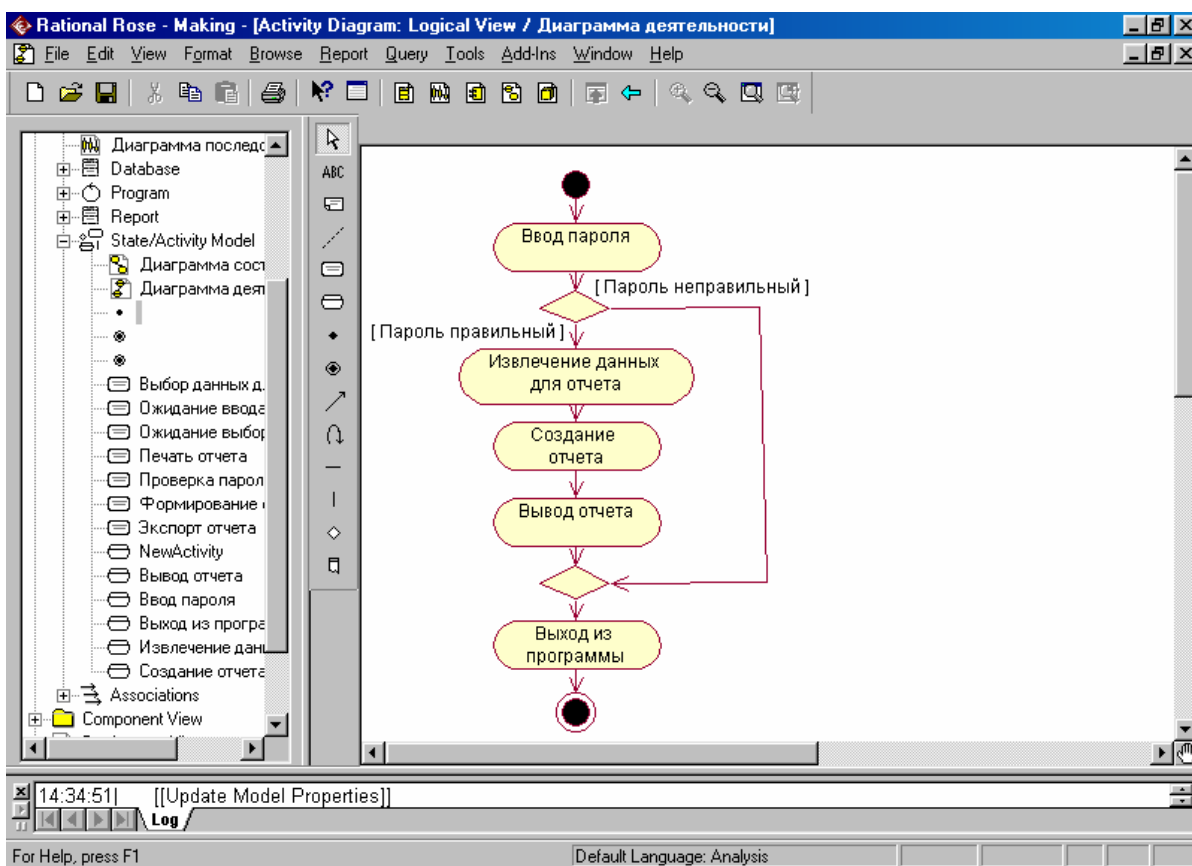


Рисунок 93 – Окончательный вид диаграммы деятельности

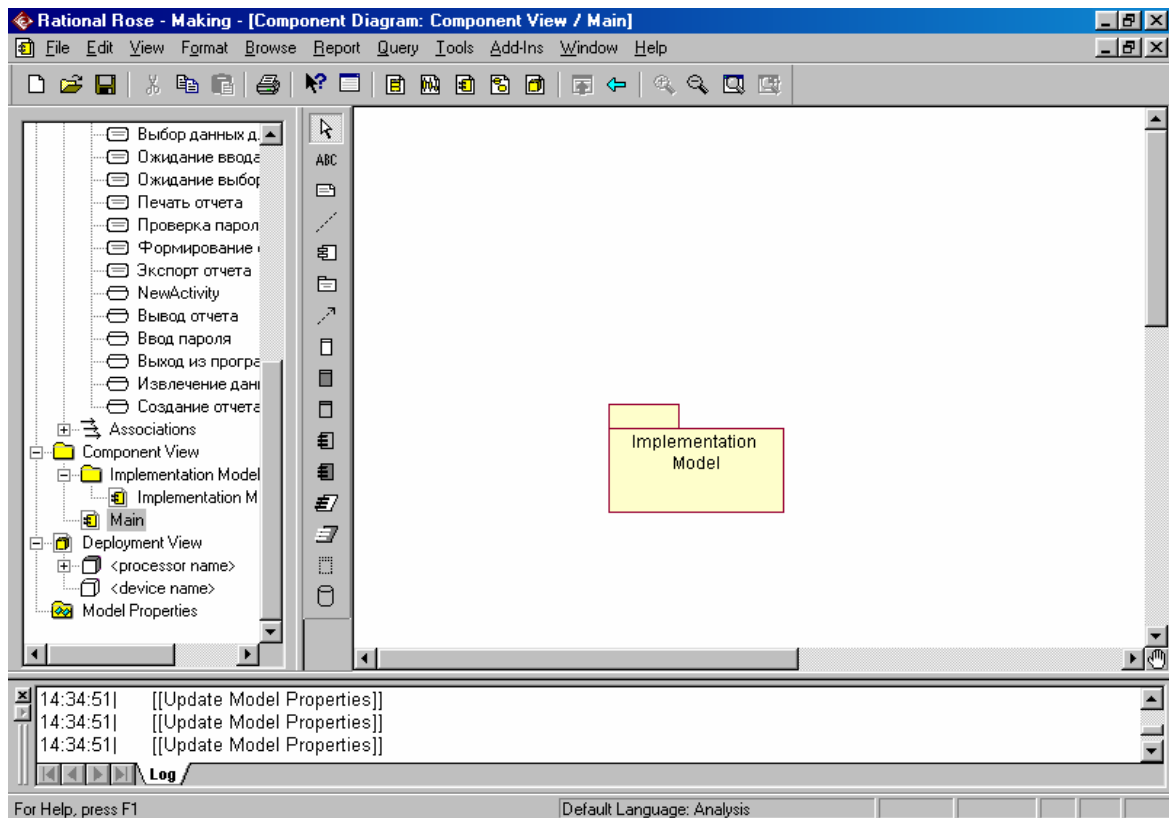


Рисунок 94 – Начальный вид диаграммы компонентов

Поместим на диаграмму новый компонент, назовем его «Главная программа» (рис. 95).

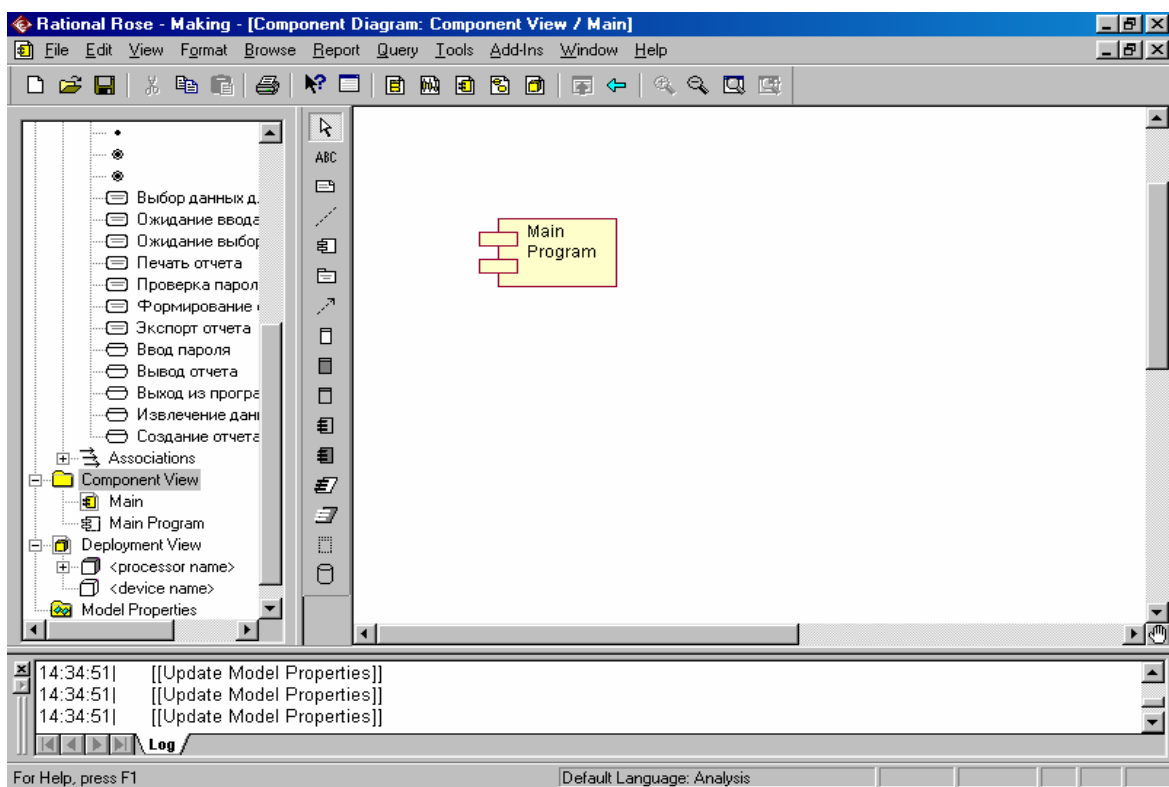


Рисунок 95 – Добавление нового компонента

Теперь можно изменить тип нового компонента: в окне спецификации выберем стереотип «EXE» (рис. 96).

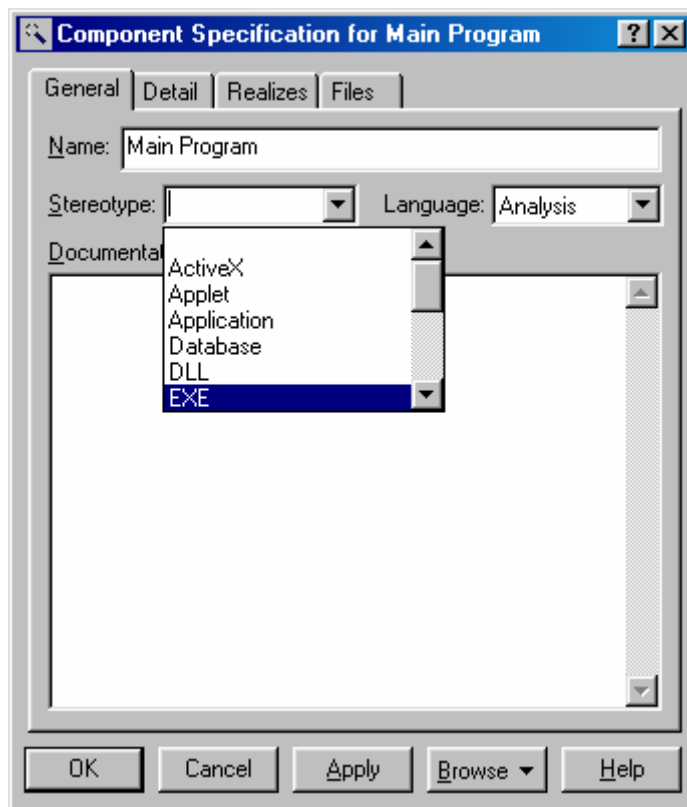


Рисунок 96 – Окно спецификации компонента

Чтобы результат изменений был явно виден на диаграмме, выберем в контекстном меню компонента пункт «Stereotype Display / Decoration» (рис. 97).

К исполняемому файлу отнесем два файла: с одной стороны, это файл Delphi-проекта (DPR), с другой – база данных. Файлу проекта можно присвоить стереотип «Main Program», изменив его изображение на «Decoration», а базу данных со стереотипом «Database» оставим в неизменном виде. Добавим связи-зависимости (dependency) между исполняемым файлом, файлом проекта и базой данной (рис. 98).

Файл проекта будет связан с тремя файлами – модулями исходных текстов программы (наличие файлов форм и тому подобное подразумевается) – Unit1.pas, Unit2.pas и Unit3.pas (рис. 99). Стереотип этих файлов можно выбрать произвольно – пусть будет Subprogram.

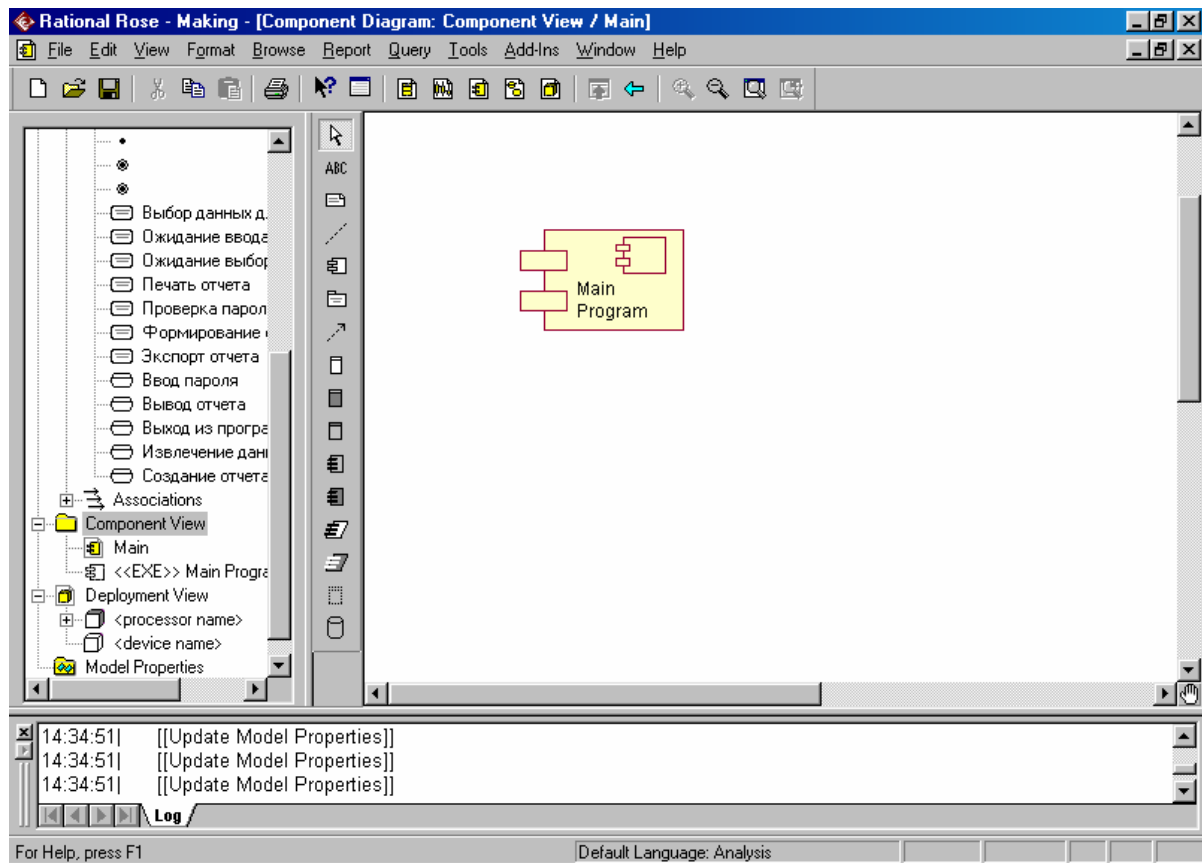


Рисунок 97 – Компонент со стереотипом «исполняемый файл»

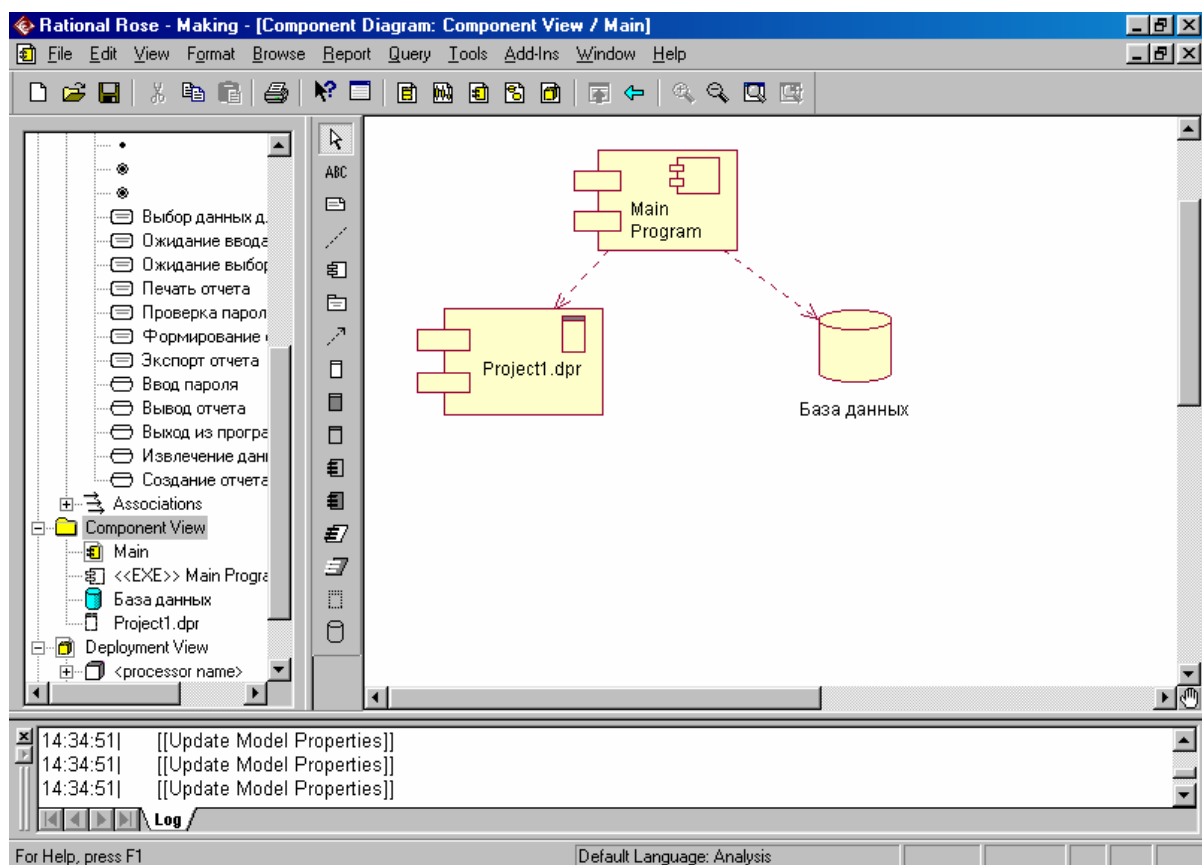


Рисунок 98 – Новые компоненты и связи между ними

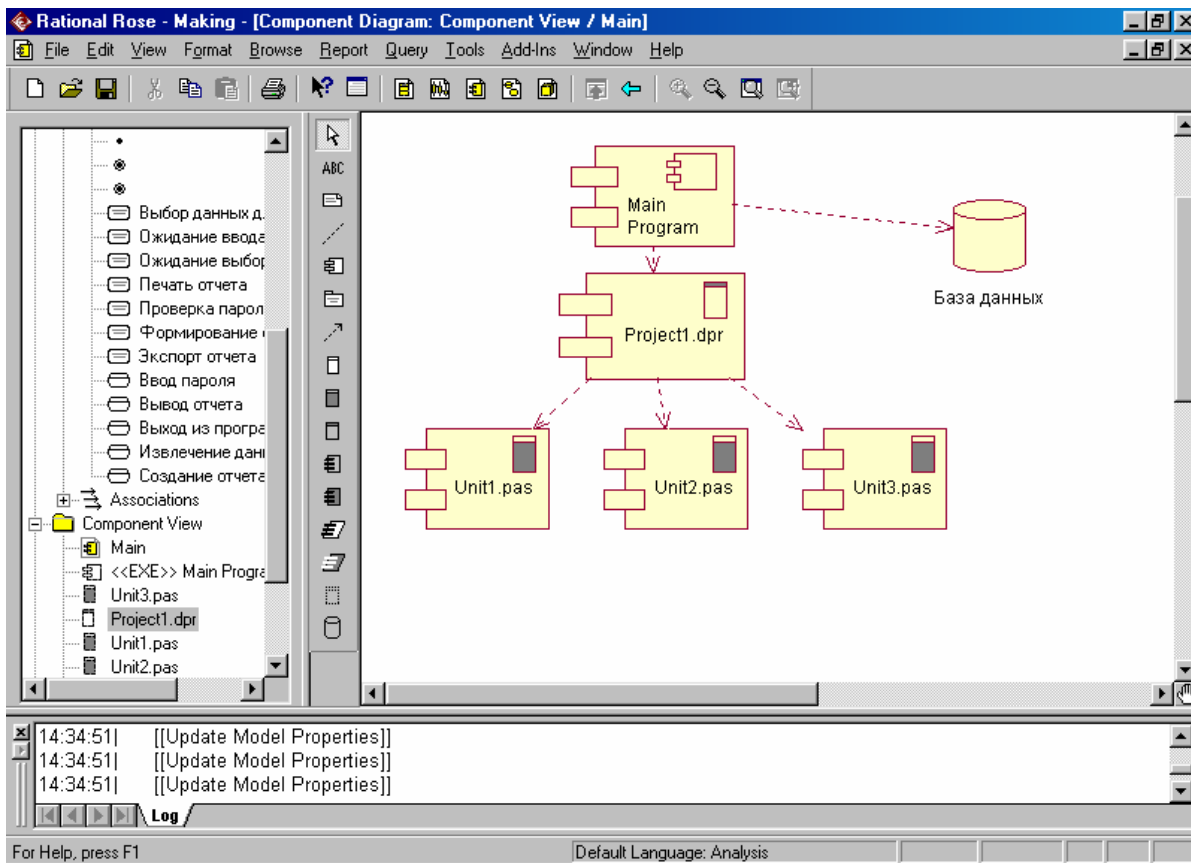


Рисунок 99 – Добавление компонентов с исходным текстом программы

Поскольку непонятно, что именно располагается в каждом из модулей, целесообразно добавить примечания, в которых уточнить этот момент. Окончательный вид диаграммы компонентов представлен на рис. 100.

Диаграмма развертывания создается пунктом меню «Browse / Deployment Diagram», в правом окне появится заготовка новой диаграммы (рис. 101). Расположенный там по умолчанию комментарий можно удалить, а два находящихся узла – оставить.

Предположим, что наша система может работать в клиент-серверном режиме. Со стороны сервера будут расположены ресурсоемкий узел – компьютер с серверной частью программы, выполняющей авторизацию пользователя, и базы данных. Со стороны клиента расположится любой компьютер, имеющий доступ к сети. В окне спецификации ресурсоемкого узла («процессора») укажем его собственное имя (например, Сервер) и стереотип – «processor», в окне спецификации узла устройства – имя «База данных» и соответствующий стереотип (рис. 102).

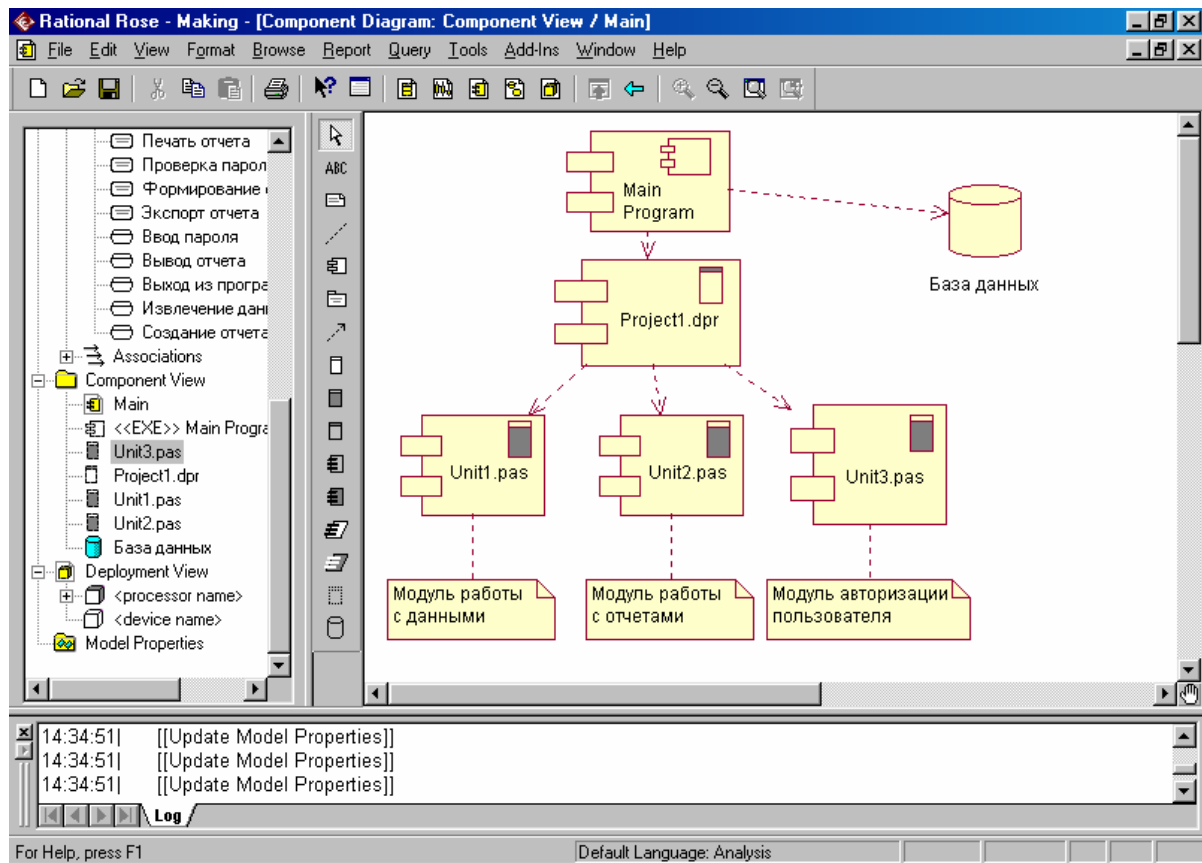


Рисунок 100 – Окончательный вид диаграммы компонентов

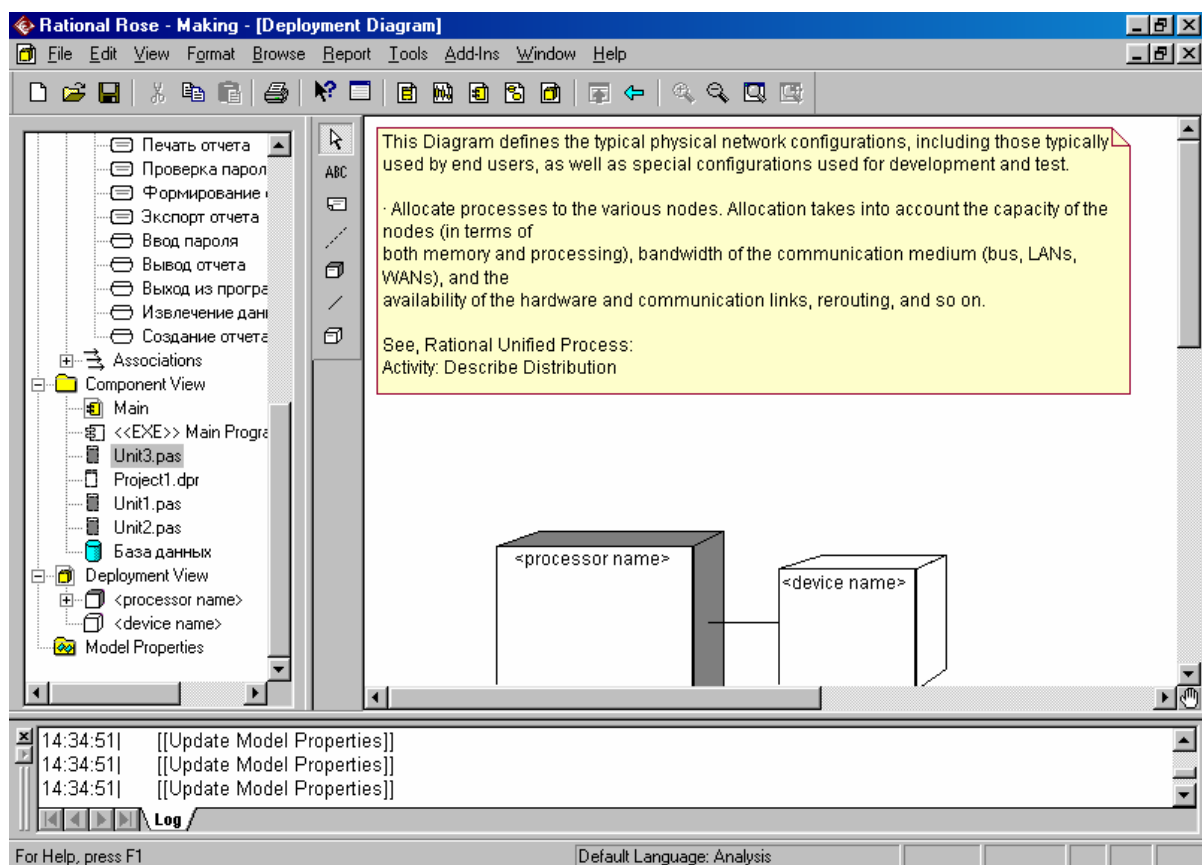


Рисунок 101 – Начальный вид диаграммы развертывания

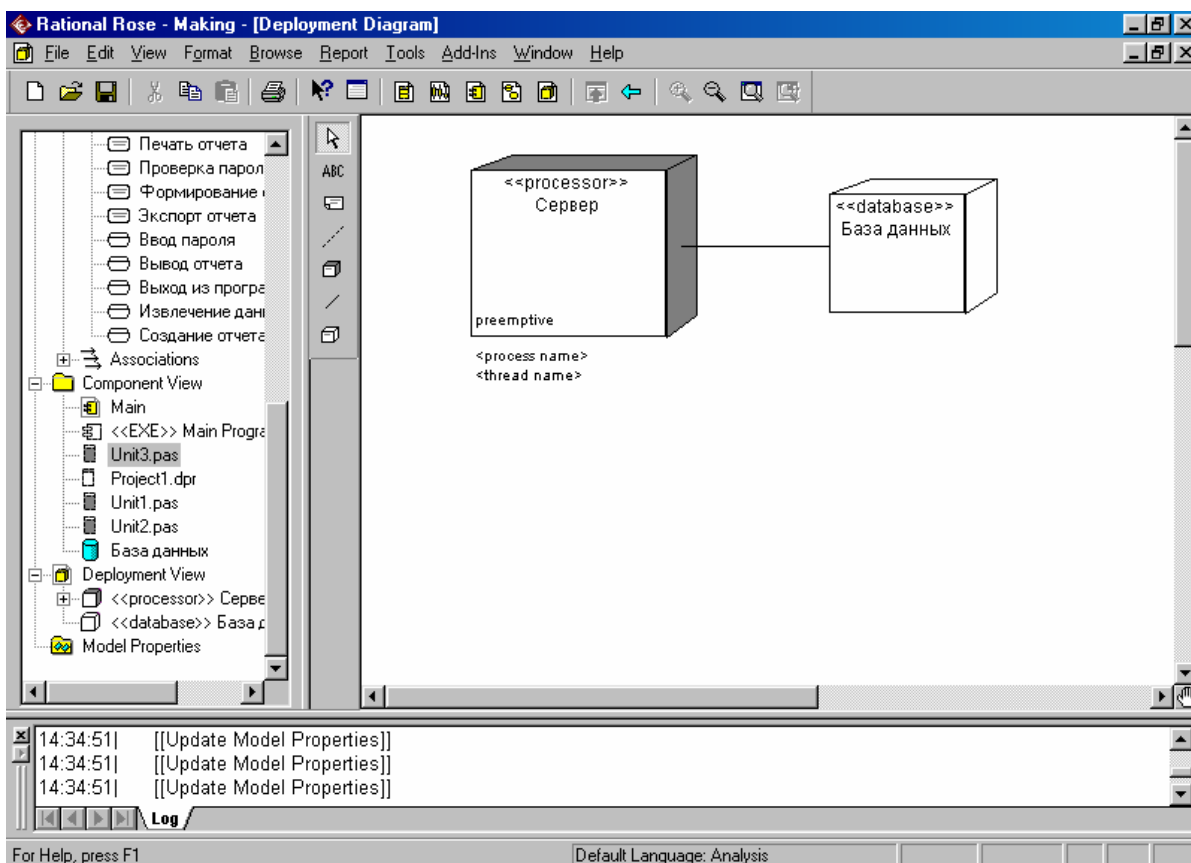


Рисунок 102 – Два узла на диаграмме развертывания

Ненужные в нашем случае строки с именами процессов или нитей управления можно убрать с экрана с помощью контекстного меню (отмена пунктов «Show Processes» и «Show Scheduling»).

Новый ресурсоемкий узел на диаграмме – клиент. Изобразим его в виде анонимного экземпляра класса «Клиент» (рис. 103), причем стереотип «processor» можно не указывать, его ресурсоемкость очевидна по внешнему виду.

Сеть, являющуюся, по сути, промежуточным устройством между серверной и клиентской частями системы, изобразим в виде обычного узла (device) со стереотипом «Net» (рис. 104).

Последние действия – размещение линий связи (connections). Предположим при этом, что доступ к базе данных может осуществляться как посредством сервера, так и непосредственно от клиента. Окончательный вид диаграммы развертывания представлен на рис. 105.

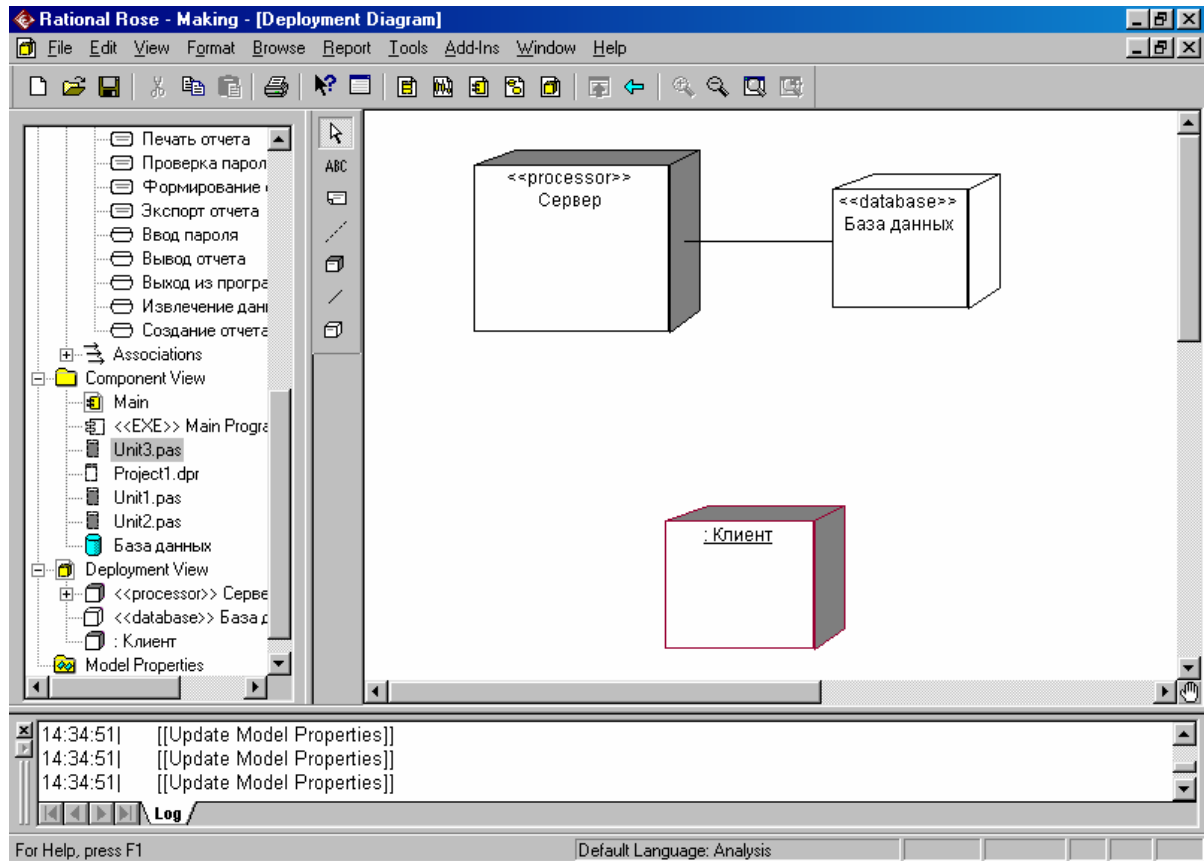


Рисунок 103 – Новый узел – анонимный экземпляр класса

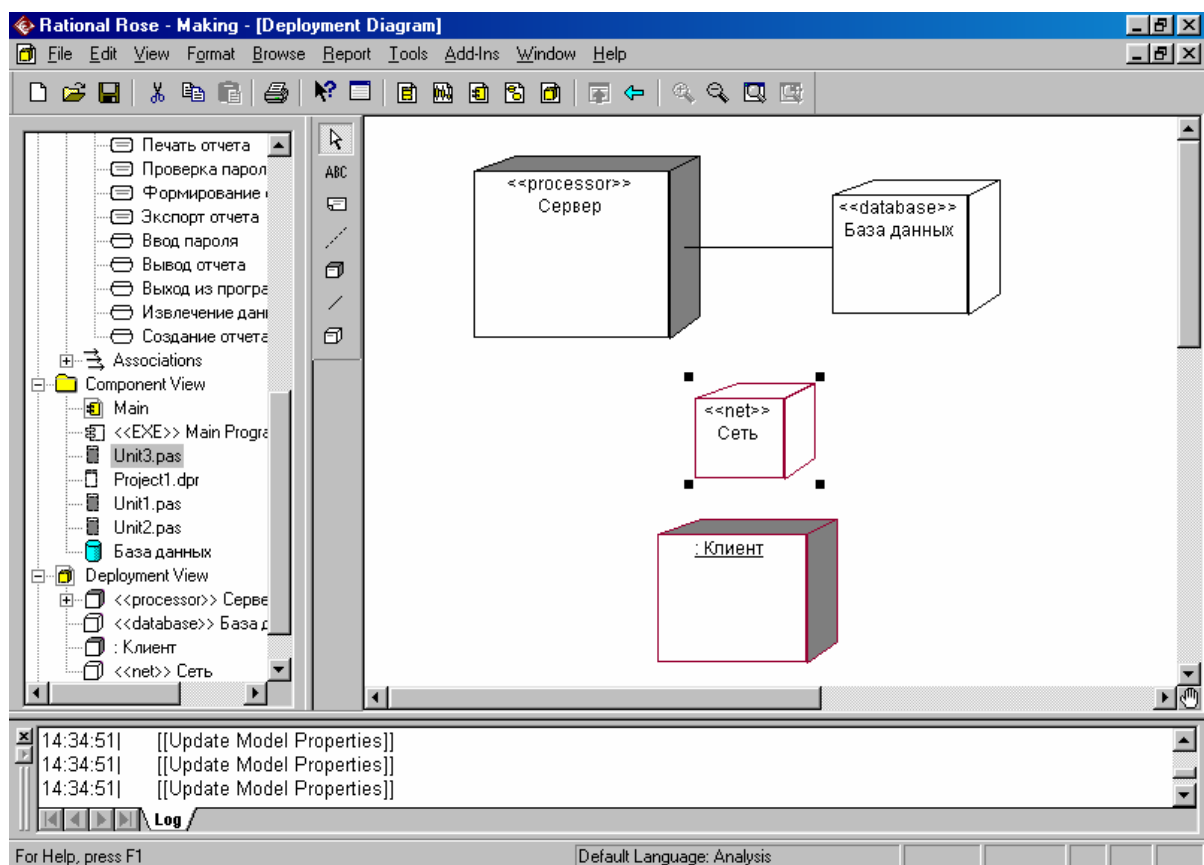


Рисунок 104 – Новое устройство «Сеть»

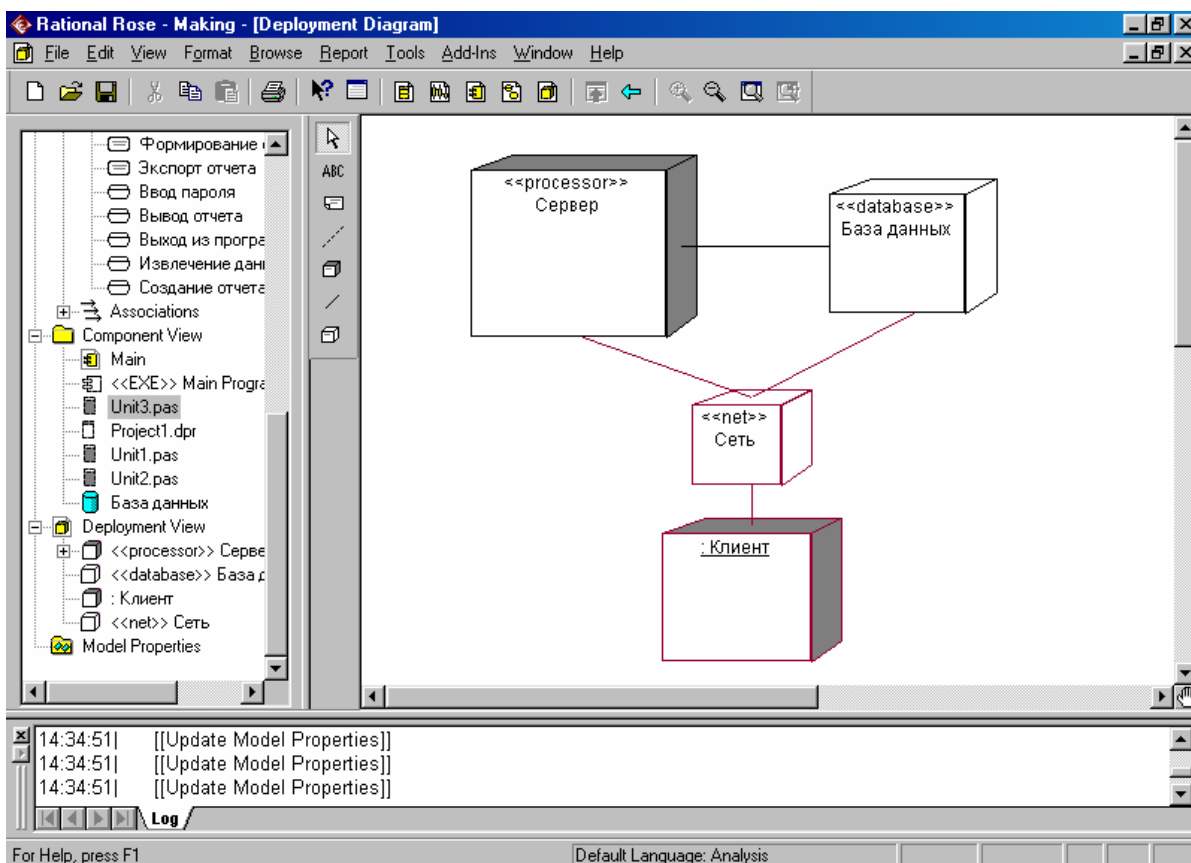


Рисунок 105 – Окончательный вид диаграммы развертывания

3.3 Генерация кода спроектированной модели в среде программирования

Созданную в среде IBM Rational Rose модель можно экспортировать в одну из сред программирования, поддерживаемых данной версией программы. Как правило, таким путем создается каркас будущей программной системы.

Перед генерацией программного кода нужно выделить все компоненты диаграммы классов и в главном меню указать язык программирования («Tools / Options / Notation / Default Language»). Затем в том же меню (пункт «Tools») выбрать этот язык и пункт «Code Generation». Результат сохраняется на диске в папке «C:\ Program Files\ Rational\ Rose\ <выбранный язык>\ source\».

В нашем случае при генерации кода на языке C++ в папке «C:\Program Files\Rational\Rose\c++\source\» будет создано 10 файлов, на-

звания и содержания которых приведены ниже (многочисленные комментарии, созданные системой, удалены).

Admin.h

```
#ifndef Admin_h
#define Admin_h 1
// Program
#include "Program.h"
class Admin
{
public:
    /// Constructors (generated)
    Admin();
    Admin(const Admin &right);
    /// Destructor (generated)
    ~Admin();
    /// Assignment Operation (generated)
    Admin & operator=(const Admin &right);
    /// Equality Operations (generated)
    int operator==(const Admin &right) const;
    int operator!=(const Admin &right) const;
    /// Get and Set Operations for Associations (generated)
    /// Association: <unnamed>%44B3DEA20104
    /// Role: Admin::<the_Program>%44B3DEA202F8
    const Program * get_the_Program () const;
    void set_the_Program (Program * value);
    // Additional Public Declarations
    /// begin Admin%44B3A68200BE.public preserve=yes
    /// end Admin%44B3A68200BE.public
protected:
    // Additional Protected Declarations
private:
    // Additional Private Declarations
```

```

private: /// implementation
    // Data Members for Associations
public: Program { -> RHN}
    Program *the_Program;
    /// end Admin::<the_Program>%44B3DEA202F8.role
    // Additional Implementation Declarations
};
// Class Admin
/// Get and Set Operations for Associations (inline)
inline const Program * Admin::get_the_Program () const
{
    return the_Program;
}
inline void Admin::set_the_Program (Program * value)
{
    the_Program = value;
}
#endif

```

Admin.cpp

```

// Admin
#include "Admin.h"
// Class Admin
Admin::Admin() { }
Admin::Admin(const Admin &right) { }
Admin::~~Admin() { }
Admin & Admin::operator=(const Admin &right) { }
int Admin::operator==(const Admin &right) const { }
int Admin::operator!=(const Admin &right) const { }
// Additional Declarations

```

Database.h

```
#ifndef Database_h
#define Database_h 1

// Program
#include "Program.h"

class Database
{
public:
    /// Constructors (generated)
    Database();
    Database(const Database &right);
    /// Destructor (generated)
    ~Database();
    /// Assignment Operation (generated)
    Database & operator=(const Database &right);
    /// Equality Operations (generated)
    int operator==(const Database &right) const;
    int operator!=(const Database &right) const;
    /// Other Operations (specified)
    void inputData ();
    /// Operation: modifyData%44B3DE2701C2
    void modifyData ();
    /// Operation: getData%44B3DE310078
    void getData ();
    /// Get and Set Operations for Associations (generated)
    const Program * get_the_Program () const;
    void set_the_Program (Program * value);
    // Additional Public Declarations
protected:
    // Additional Protected Declarations
private:
    /// Get and Set Operations for Class Attributes (generated)
    const void get_data () const;
```

```

    void set_data (void value);
// Additional Private Declarations
    /// begin Database%44B3DDF60082.private preserve=yes
    /// end Database%44B3DDF60082.private
private: /// implementation
    // Data Members for Class Attributes
    void data;
    // Data Members for Associations
public: Program { -> RHN}
    Program *the_Program;
    // Additional Implementation Declarations
};
// Class Database
/// Get and Set Operations for Class Attributes (inline)
inline const void Database::get_data () const
{
    return data;
}
inline void Database::set_data (void value)
{
    data = value;
}
/// Get and Set Operations for Associations (inline)
inline const Program * Database::get_the_Program () const
{
    return the_Program;
}
inline void Database::set_the_Program (Program * value)
{
    the_Program = value;
}
#endif

```


Database.cpp

```
// Database
#include "Database.h"
// Class Database
Database::Database() { }
Database::Database(const Database &right) { }
Database::~~Database() { }
Database & Database::operator=(const Database &right) { }
int Database::operator==(const Database &right) const { }
int Database::operator!=(const Database &right) const { }
///## Other Operations (implementation)
void Database::inputData () { }
void Database::modifyData () { }
void Database::getData () { }
// Additional Declarations
```

Program.h

```
#ifndef Program_h
#define Program_h 1
// User
#include "User.h"
// Report
#include "Report.h"
// Database
#include "Database.h"
// Admin
#include "Admin.h"
class Program
{
public:
    ///## Constructors (generated)
    Program();
    Program(const Program &right);
```

```

    /// Destructor (generated)
    ~Program();
    /// Assignment Operation (generated)
    Program & operator=(const Program &right);
    /// Equality Operations (generated)
    int operator==(const Program &right) const;
    int operator!=(const Program &right) const;
    /// Other Operations (specified)
    Boolean authorization ();
    /// Get and Set Operations for Associations (generated)
    const Admin * get_the_Admin () const;
    void set_the_Admin (Admin * value);
    const User * get_the_User () const;
    void set_the_User (User * value);
    const Database * get_the_Database () const;
    void set_the_Database (Database * value);
    const Report * get_the_Report () const;
    void set_the_Report (Report * value);
    // Additional Public Declarations
protected:
    // Additional Protected Declarations
private:
    // Additional Private Declarations
private: /// implementation
    // Data Members for Associations

    Admin *the_Admin;
    /// Association: <unnamed>%44B3DEA40046
lic: User { -> RHN}
    User *the_User;
public: Database { -> RHN}
    Database *the_Database;
public: Report { -> RHN}

```

```

    Report *the_Report;
// Additional Implementation Declarations
};
// Class Program
///## Get and Set Operations for Associations (inline)
inline const Admin * Program::get_the_Admin () const
{
    return the_Admin;
}
inline void Program::set_the_Admin (Admin * value)
{
    the_Admin = value;
}
inline const User * Program::get_the_User () const
{
    return the_User;
}
inline void Program::set_the_User (User * value)
{
    the_User = value;
}
inline const Database * Program::get_the_Database () const
{
    return the_Database;
}
inline void Program::set_the_Database (Database * value)
{
    the_Database = value;
}
inline const Report * Program::get_the_Report () const
{
    return the_Report;
}

```

```

inline void Program::set_the_Report (Report * value)
{
    the_Report = value;
}
#endif

```

Program.cpp

```

// Program
#include "Program.h"
// Class Program
Program::Program() { }
Program::Program(const Program &right) { }
Program::~~Program() { }
Program & Program::operator=(const Program &right) { }
int Program::operator==(const Program &right) const { }
int Program::operator!=(const Program &right) const { }
///## Other Operations (implementation)
Boolean Program::autorization () { }
// Additional Declarations

```

Report.h

```

#ifndef Report_h
#define Report_h 1
// Program
#include "Program.h"
class Report
{
public:
    ///## Constructors (generated)
    Report();
    Report(const Report &right);
    ///## Destructor (generated)
    ~Report();

```

```

    /// Assignment Operation (generated)
    Report & operator=(const Report &right);
    /// Equality Operations (generated)
    int operator==(const Report &right) const;
    int operator!=(const Report &right) const;
    /// Other Operations (specified)
    void forming ();
    void print ();
    void export ();
    /// Get and Set Operations for Associations (generated)
    const Program * get_the_Program () const;
    void set_the_Program (Program * value);
    // Additional Public Declarations
protected:
    // Additional Protected Declarations
private:
    /// Get and Set Operations for Class Attributes (generated)
    const void get_data () const;
    void set_data (void value);
    // Additional Private Declarations
private: /// implementation
    // Data Members for Class Attributes
    void data;
    // Data Members for Associations
    Program *the_Program;
};
// Class Report
/// Get and Set Operations for Class Attributes (inline)
inline const void Report::get_data () const
{
    return data;
}
inline void Report::set_data (void value)

```

```

{
    data = value;
}

///  

inline const Program * Report::get_the_Program () const
{
    return the_Program;
}

inline void Report::set_the_Program (Program * value)
{
    the_Program = value;
}

#endif

```

Report.cpp

```

// Report
#include "Report.h"

// Class Report
Report::Report() { }
Report::Report(const Report &right) { }
Report::~~Report() { }
Report & Report::operator=(const Report &right) { }
int Report::operator==(const Report &right) const { }
int Report::operator!=(const Report &right) const { }

///  

void Report::forming () { }
void Report::print () { }
void Report::export () { }

// Additional Declarations

```

User.h

```

#ifndef User_h
#define User_h 1

// Program

```

```

#include "Program.h"
class User
{
public:
    /// Constructors (generated)
    User();
    User(const User &right);
    /// Destructor (generated)
    ~User();
    /// Assignment Operation (generated)
    User & operator=(const User &right);
    /// Equality Operations (generated)
    int operator==(const User &right) const;
    int operator!=(const User &right) const;
    /// Get and Set Operations for Associations (generated)
    const Program * get_the_Program () const;
    void set_the_Program (Program * value);
    // Additional Public Declarations
protected:
    // Additional Protected Declarations
private:
    // Additional Private Declarations
private: /// implementation
    // Data Members for Associations
    Program *the_Program;
    // Additional Implementation Declarations
};
// Class User
/// Get and Set Operations for Associations (inline)

inline const Program * User::get_the_Program () const
{
    return the_Program;
}

```

```

}
inline void User::set_the_Program (Program * value)
{
    the_Program = value;
}
#endif

```

User.cpp

```

// User
#include "User.h"
// Class User
User::User() { }
User::User(const User &right) { }
User::~~User() { }
User & User::operator=(const User &right) { }
int User::operator==(const User &right) const { }
int User::operator!=(const User &right) const { }
// Additional Declarations

```

4 ПРИМЕРЫ ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

В этом разделе представлены примеры спроектированных информационных систем для решения определенных экономических и управленческих задач. Все представленные модели разработаны студентами специальности «Экономическая кибернетика» в рамках курсового и дипломного проектирования под руководством или с участием автора пособия. Часть моделей создавалась при помощи IBM Rational Rose, часть – с использованием других CASE-средств или просто «вручную» (в MS-Word); все модели приводятся здесь в «первозданном» варианте в целях иллюстрации многообразия представления UML-моделей.

Каждый подраздел, помимо собственно модели на языке UML, содержит краткое описание предметной области и имеющихся в ней проблем, представляя, по сути, научную статью.

4.1 Информационная система для функционирования кадрового агентства

За последнее время кадровый бизнес в нашей стране превратился в бурно развивающуюся сферу экономики. И если раньше все функции кадрового менеджмента возлагались на соответствующие службы внутри предприятий, то сейчас широкий спектр услуг по подбору кадров предоставляют специализированные кадровые агентства. С учетом специфики работы подобных организаций, где циркулируют большие объемы информации, организация работы с данными здесь должна быть реализована наиболее эффективно и в наиболее удобном для пользователей варианте.

Анализ информационного обеспечения кадровых агентств показал, что в большинстве случаев используются стандартные приложения для обработки баз данных, позволяющие осуществлять ввод, хранение и модификацию данных, а также поиск в базе данных. В то же время многообразие всевозможных требований приводит к решению о необходимости создания специализированной информационной системы для функционирования кадрового агентства.

По результатам анализа предметной области, касающейся конкретного предприятия (кадрового агентства Краматорского отделения Донецкой торгово-промышленной палаты), была поставлена задача реализации следующих оптимизационных решений.

Во-первых, усовершенствование задачи поиска (он должен быть двунаправленным и многопараметрическим):

- поиск вариантов, удовлетворяющих требованиям по указанной вакансии в таблице соискателей (условие поиска может включать обязательные требования (пол, возраст, специальность, уровень квалификации, уровень образования, опыт работы, наличие собственного авто, телефона и т.д., причем некоторые из этих полей вообще могут не вклю-

чатся в поиск, как не имеющие значения) и необязательные (регион, город работы, заработная плата и т.п.);

- поиск возможных вариантов трудоустройства для клиента-соискателя в таблице имеющихся вакансий (условием поиска являются данные о конкретном клиенте-соискателе, которые заносятся из заполняемых анкет, т.е. уровень образования, специальность, квалификация, опыт работы, желаемая должность, личные данные и т.д.);

Во-вторых, добавление модуля формирования статистики.

- учет статистики выполненных заказов. После отбора подходящих вариантов к работодателю направляются претенденты на должность, и в таблицу учета статистики заносятся данные о том, принят претендент или нет, т.е. закрыта вакансия или нет. Ведение статистики дает возможность составления итоговой отчетности по конкретному клиенту-работодателю (какие претенденты были предложены и какие приняты) и клиенту-соискателю (какие вакансии были предложены и какие приняты);

- составление отчетности за период по удовлетворенным и не удовлетворенным заявкам с определением тенденции и анализом деятельности.

Применение унифицированного языка визуального моделирования UML позволило всесторонне рассмотреть и представить систему в последовательности от наиболее общей и абстрактной концептуальной модели к логической, а затем и к физической модели.

Логические аспекты статического представления системы были представлены при помощи диаграммы классов (рис. 106).

Особенности реализации операций классов были учтены и рассмотрены при помощи диаграммы деятельности (рис. 107).

Функциональные характеристики системы были представлены в диаграмме вариантов использования (рис. 108).

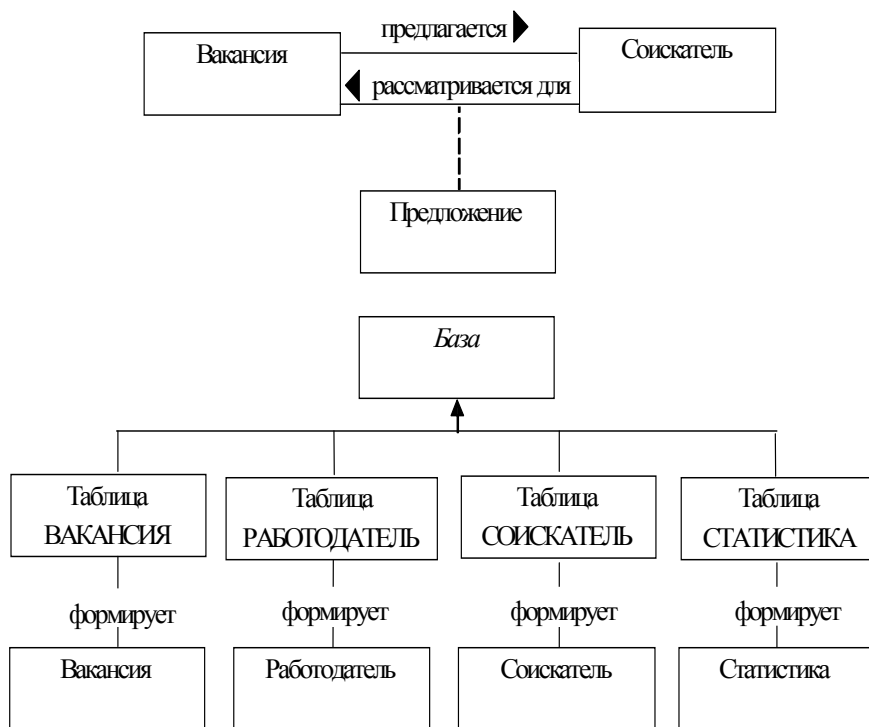
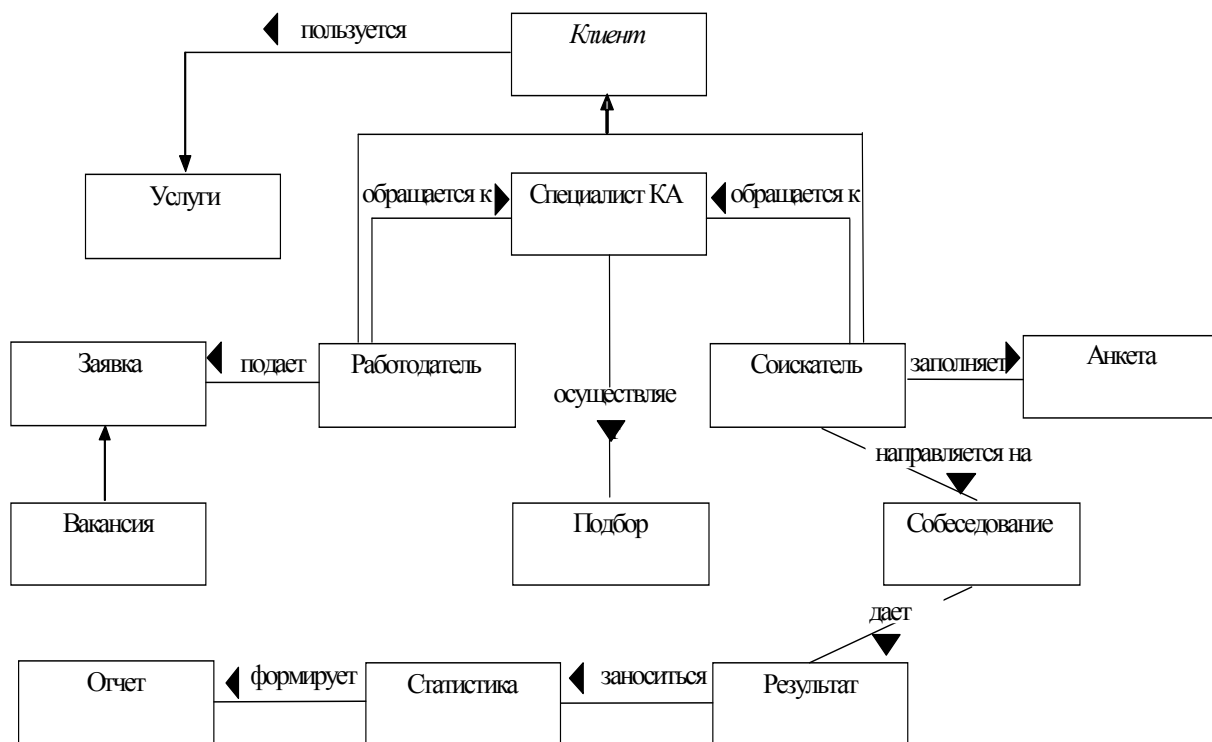


Рисунок 106 – Детализированная диаграмма классов

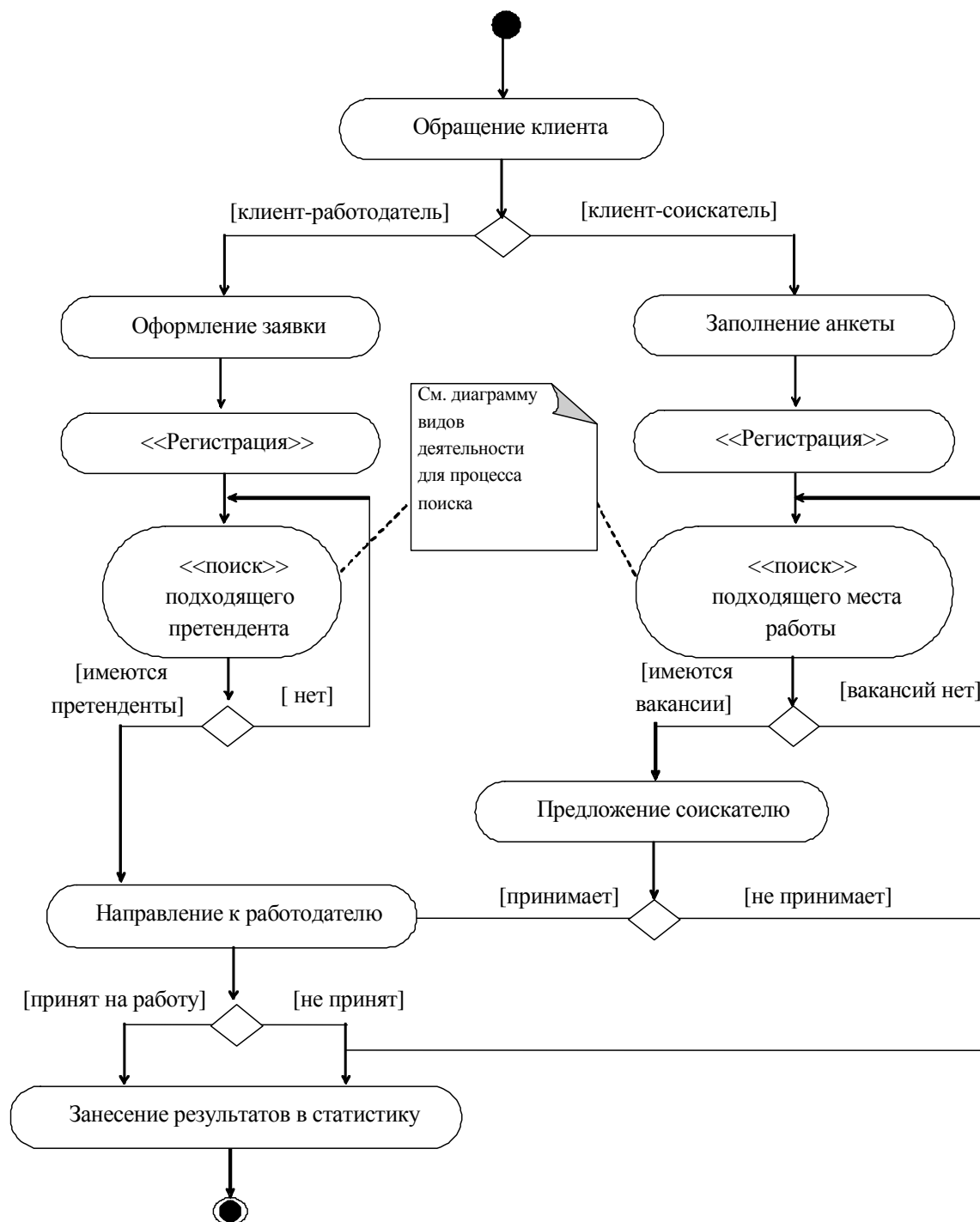


Рисунок 107 – Диаграмма деятельности для процесса работы с клиентами

Взаимодействие элементов системы, рассматриваемое в информационном аспекте их коммуникации, было рассмотрено при помощи соответствующих диаграмм взаимодействия: диаграммы последовательностей (временные аспекты взаимодействия – рис. 109) и диаграммы кооперации (структурные особенности взаимодействия – рис. 110).

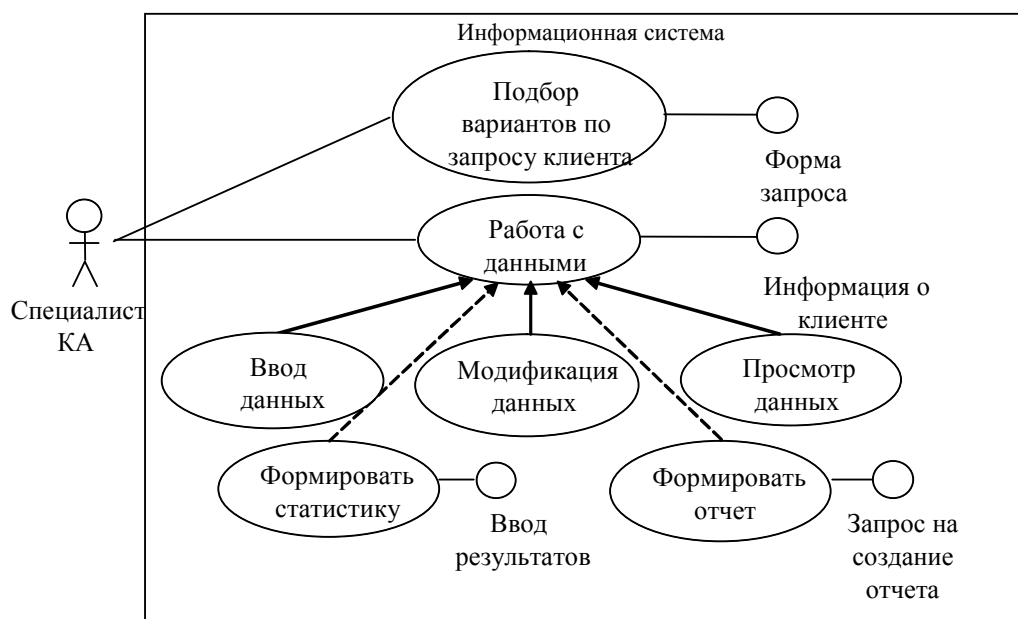


Рисунок 108 – Диаграмма вариантов использования

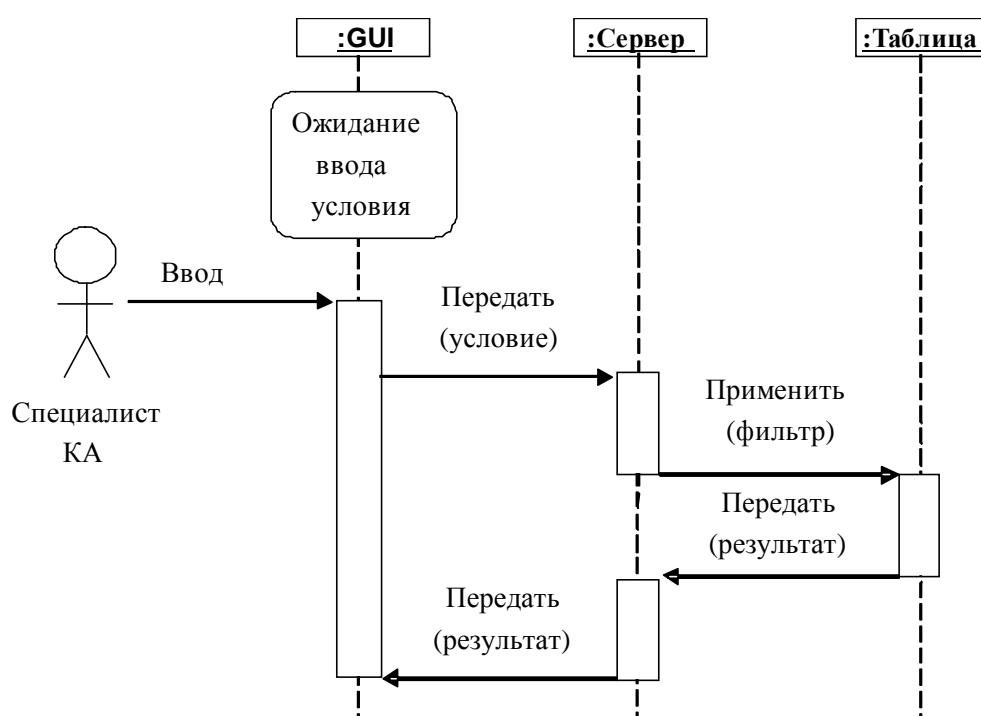


Рисунок 109 – Диаграмма последовательностей для процесса подбора вариантов (Поиск)

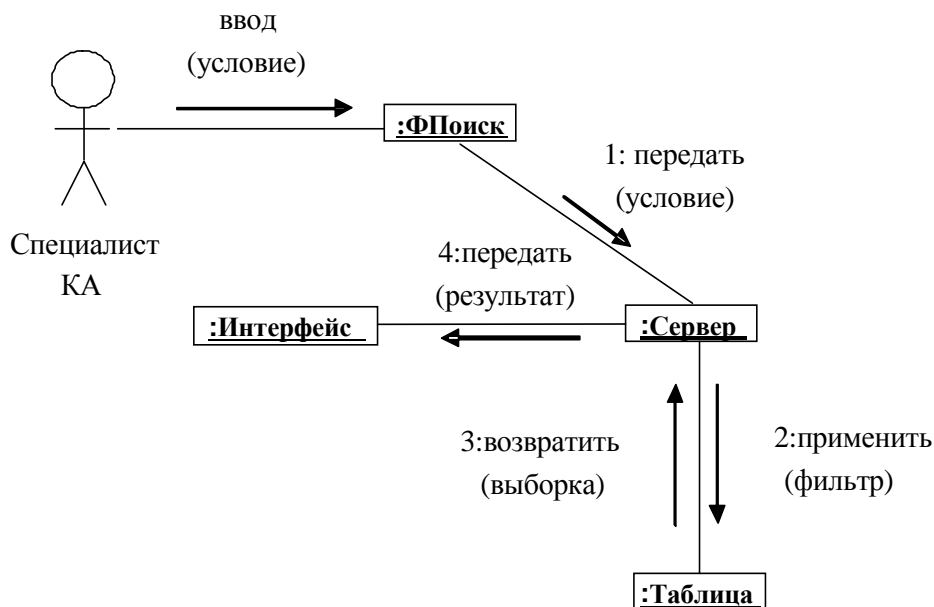


Рисунок 110 – Диаграмма коопераций для процесса подбора вариантов (Поиск)

Логическое представление системы позволило провести анализ структурных и функциональных отношений между элементами модели системы. Для создания конкретной физической системы, где элементы логического представления должны быть реализованы в конкретные материальные сущности, использовались диаграммы компонентов (рис. 111) и развертывания (рис. 112) с графическим изображением процессоров, устройств, процессов и связей между ними.

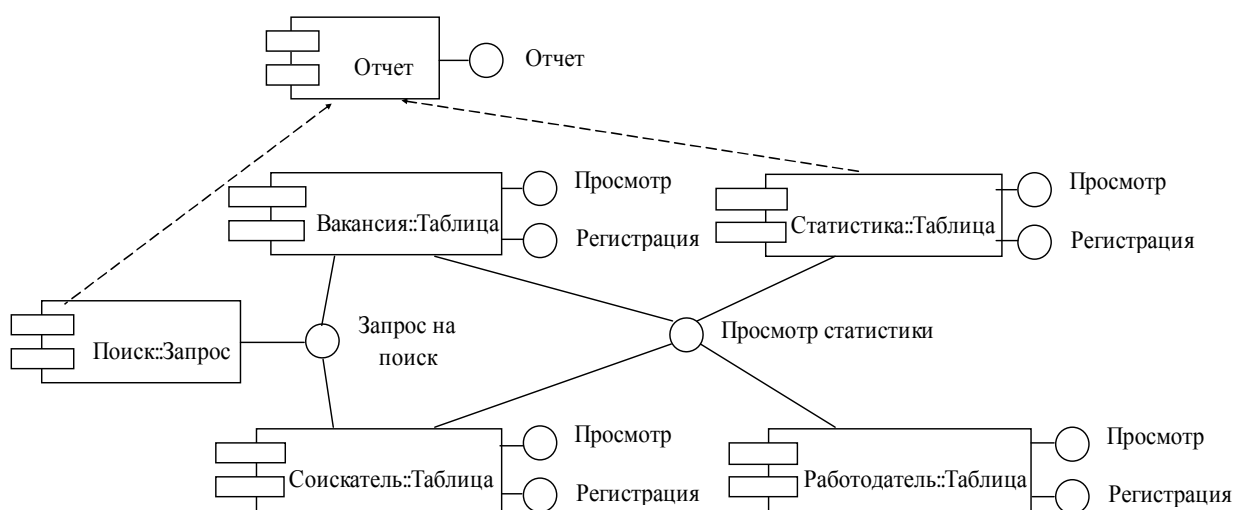


Рисунок 111 – Диаграмма компонентов информационной системы

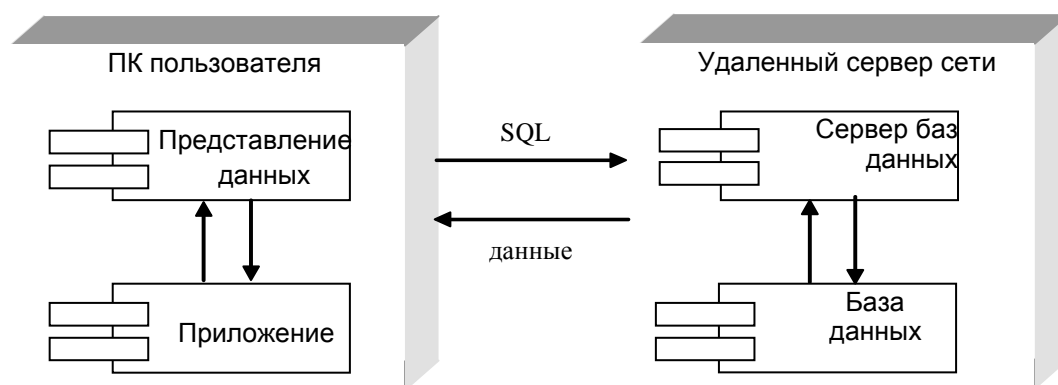


Рисунок 112 – Диаграмма развертывания информационной системы

Программная реализация разработанной модели была выполнена в среде Borland Delphi 6. Разработанный программный продукт на основе построенной модели позволяет повысить оперативность и качество обслуживания клиентов за счет применения многопараметрического двунаправленного поиска (подбора вариантов). Ведение статистических данных о результатах деятельности кадрового агентства по подбору специалистов и формирование соответствующих отчетов позволяет во многом сократить объемы рутинной работы и в целом повысить продуктивность труда работников кадрового агентства. Результаты анализа эффективности внедрения разработанного программного продукта показывают, что его внедрение позволит сэкономить 2644 грн. в год за счет высвобождения рабочего времени специалиста кадрового агентства и в целом повысить качество предоставляемых услуг [9-13].

4.2 Информационная система для автоматизированного составления расписания занятий в высшем учебном заведении

Каждый учебный семестр любого высшего учебного заведения начинается с составления расписания занятий. Как правило, при этом используется человеческий фактор без применения средств автоматизации. Однако каким бы опытным не был диспетчер, этот процесс является очень трудоемким; человек не в состоянии справиться с таким количест-

вом информации, не допустив при этом ни одной ошибки. Без сомнения, это негативно отражается на процессе обучения.

К настоящему времени разработан ряд компьютерных программ для автоматического составления расписания. **CyberMatrix Class Scheduler** может использоваться как одним преподавателем, для составления личного расписания, так и диспетчерским отделом для составления расписания занятий всего учебного заведения. В программе присутствуют средства фильтрации и поиска; ее недостатком является наличие только английского интерфейса, что однозначно вызовет затруднения при работе. «**Моделедром – Расписание**» в качестве исходных данных использует список дисциплин, список учебных групп и учебную нагрузку каждой группы; позволяет печатать полученные таблицы расписаний с разбивкой по листам. К сожалению, демо-версия не дает возможность оценить уровень программы и целесообразность ее приобретения за немалую цену. **Расписание ПРО** предназначена для составления расписания занятий любых учебных заведений в ручном и автоматическом режиме; в любой момент в расписание можно внести изменения, распечатать его или экспортировать таблицу в MS-Excel. Программа позволяет вести списки кабинетов, преподавателей, предметов, групп (классов), задавать связи между ними и нагрузку по предметам; есть возможность задать рабочее время индивидуально для преподавателя и приоритет предметов в расписании. Предусматривается как ручное составление расписания (с отслеживанием заданных ограничений), так и автоматический расчет – с дальнейшей «ручной» корректировкой (так как не существует строгих алгоритмов составления расписания, в автоматическом режиме предлагаются лишь возможные варианты автоматического расчета). Является универсальным средством составления расписаний, однако достаточно сложна и неудобна в обращении. «**Ректор**» также позволяет составлять расписания для любого учебного заведения и предусматривает ручной и автоматический режимы работы; составленное расписание может быть выведено в файл форматов Word 97/2000, Excel 97/2000, HTML 4.0 с разбиением на страницы. Однако, несмотря на заявленную универсальность, данная программа реально подготовлена для работы исключительно в средних учебных заведениях: все термины («урок», «ученики» и т.д.)

«жестко защиты» в программу и не предусматривают модификации. Таким образом, изучив имеющиеся программы, можно сделать вывод, что число разработанных программ относительно невелико, а качество их выполнения далеко от совершенства. Лидером в этой области является ООО «Дигси», создавшее программу «Расписание Про» и представившее условно-бесплатную (ShareWare) версию.

Логичным выводом является решение спроектировать собственную систему для автоматизированного составления расписания. Основные требования к системе формулируются следующим образом:

- осуществлять как «ручное» составление расписания, так и автоматическое с возможностью дальнейшей «ручной» корректировки;
- предполагать возможность внесения изменений в составленное расписание (замена предметов, преподавателей, аудиторий);
- производить учет выполнения нагрузки;
- распечатывать расписания по курсам, группам, кафедрам и другую отчетность (вывод в Excel).

Предполагается, что система в качестве исходных данных должна использовать аудиторный фонд, распределение нагрузки по кафедрам (группам) и распределение нагрузки по преподавателям внутри кафедры, т.е. исходная информация для диспетчеров и для программы идентична и не потребует дополнительной обработки. При этом программа должна быть легко освоена даже не знакомым с компьютерной техникой персоналом, а составление новых или редактирование существующих расписаний занимало бы не более часа. Каждый вариант расписания должен рассматриваться как отдельный проект; введя однажды исходные данные, можно будет в дальнейшем делать копию проекта и работать с ней.

Из-за достаточной сложности проектируемой информационной системы обычный структурный подход не подходит, так как при нем основой системы является алгоритм – последовательность действий по решению задач. Запись процесса составления расписания в виде алгоритма нецелесообразна, и мы применим объектно-ориентированный подход. Информационная система в таком случае будет представлять собой совокупность взаимосвязанных объектов.

Разработка информационной системы проводится в три этапа:

1. Объектно-ориентированный анализ предметной области: определение ключевых абстракций, идентификация классов и объектов.
2. Концептуальное, логическое и физическое моделирование информационной системы; построение соответствующих диаграмм.
3. Программная реализация разработанной модели.

Первый этап включает анализ действий по составлению расписания, выделение активных и пассивных классов (объектов). Актером (активным объектом) здесь является диспетчер, пассивными объектами – таблицы, предоставляемые кафедрами согласно учебным планам, и аудиторный фонд.

На втором этапе создается информационная модель проектируемой системы, для чего используется унифицированный язык моделирования UML. Сначала формулируются требования к разрабатываемой системе, определяются функции, которые она должна реализовать, и задачи, которые она должна решать. На основе этих данных формируется диаграмма вариантов использования (рис.113).

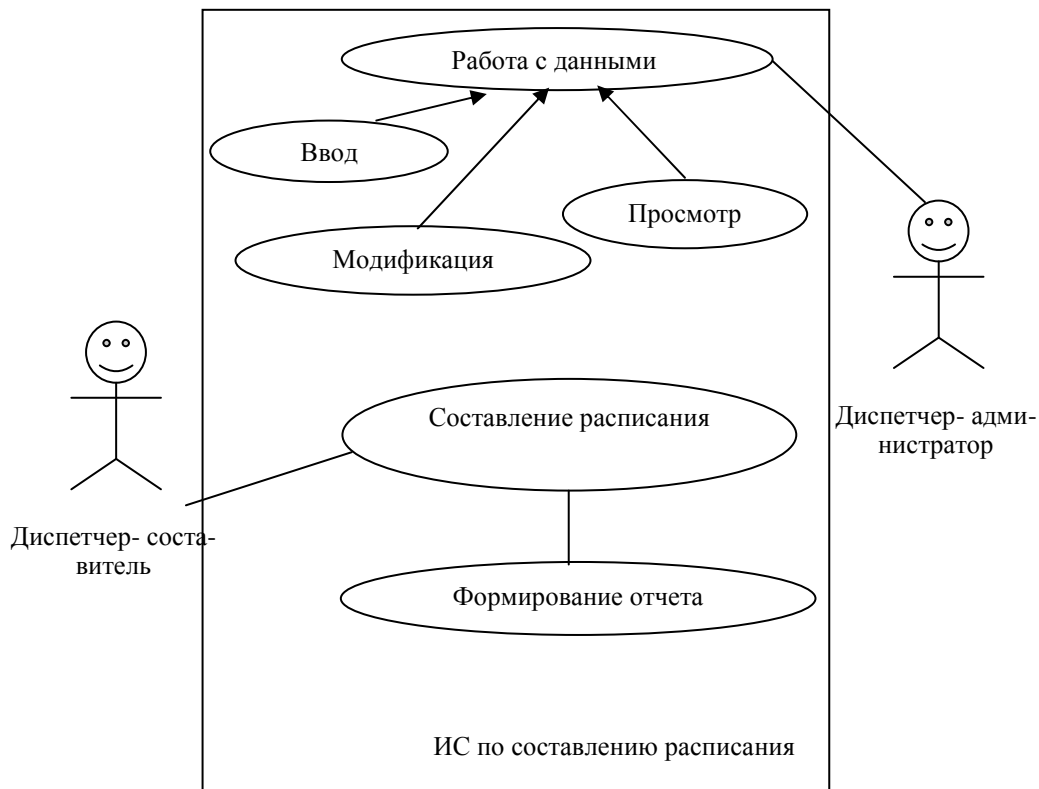


Рисунок 113 – Диаграмма вариантов использования

В процессе работы с системой будут работать два актера (активных объекта), один из которых будет заниматься работой с данными «диспетчер-администратор», а второй – собственно составлением расписания и всевозможных отчетов (диспетчер-составитель). Предполагается наличие таких вариантов использования системы: составление расписания, формирование отчета и работа с данными, которая предполагает ввод, модификацию и просмотр данных.

Далее строится структурно-логическая схема системы в виде диаграммы классов (рис.114).

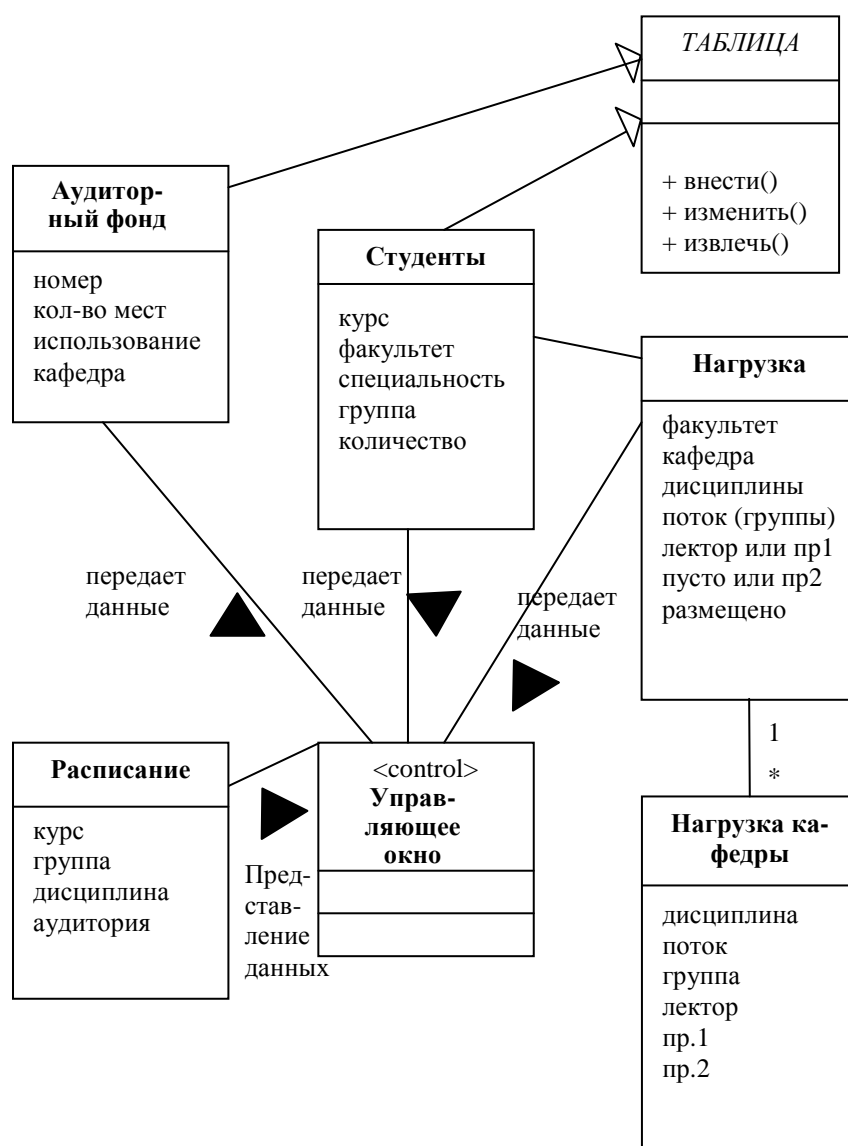


Рисунок 114 – Диаграмма классов

Эта диаграмма служит для представления статической структуры модели системы, различных взаимосвязей между отдельными сущностями-

ми предметной области, такими как объекты и подсистемы, а также описания их внутренней структуры и типов отношений.

Мы предполагаем существование в системе следующих классов: аудиторный фонд, студенты, нагрузка, расписание, управляющее окно и таблица. Класс «Таблица» является абстрактным: он не имеет экземпляров и связан с классами «Аудиторный фонд», «Студенты» и «Нагрузка» отношением наследования. Стереотип «control» класса «Управляющее окно» указывает на то, что этот класс отвечает за координацию действий других классов.

Динамические аспекты поведения системы (собственно процесса составления расписания) отображаются на диаграммах кооперации и последовательности. На диаграмме кооперации поведение системы описывается на уровне отдельных объектов, которые обмениваются между собой сообщениями, чтобы достичь нужной цели или реализовать некоторый вариант использования (рис. 115).

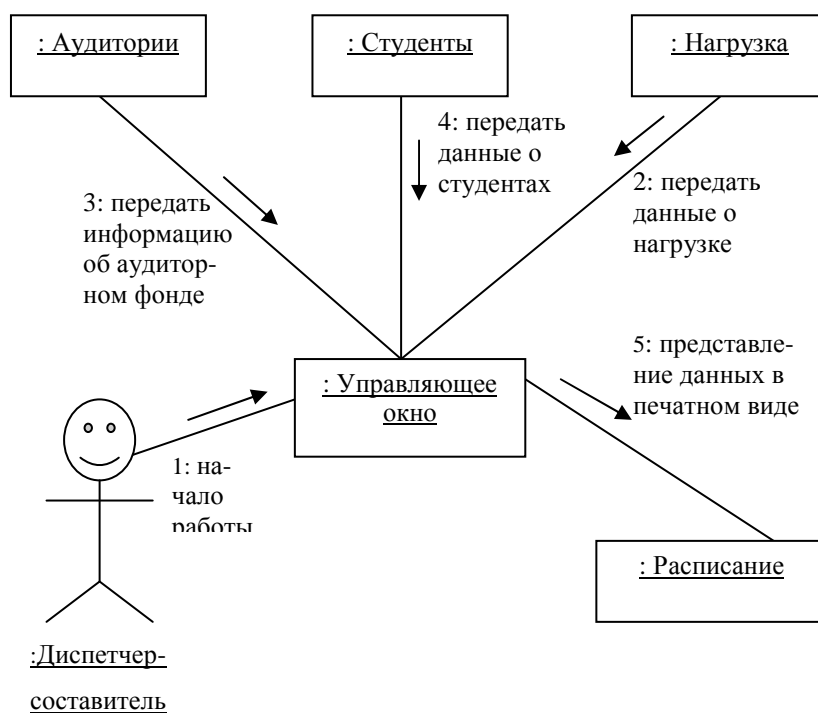


Рисунок 115 – Диаграмма коопераций

С помощью диаграмм последовательности можно описать полный контекст взаимодействий как своеобразный временной график «жизни» всей совокупности объектов, взаимодействующих между собой для реализации варианта использования программной системы, достижения бизнес-цели или выполнения какой-либо задачи (рис. 116).

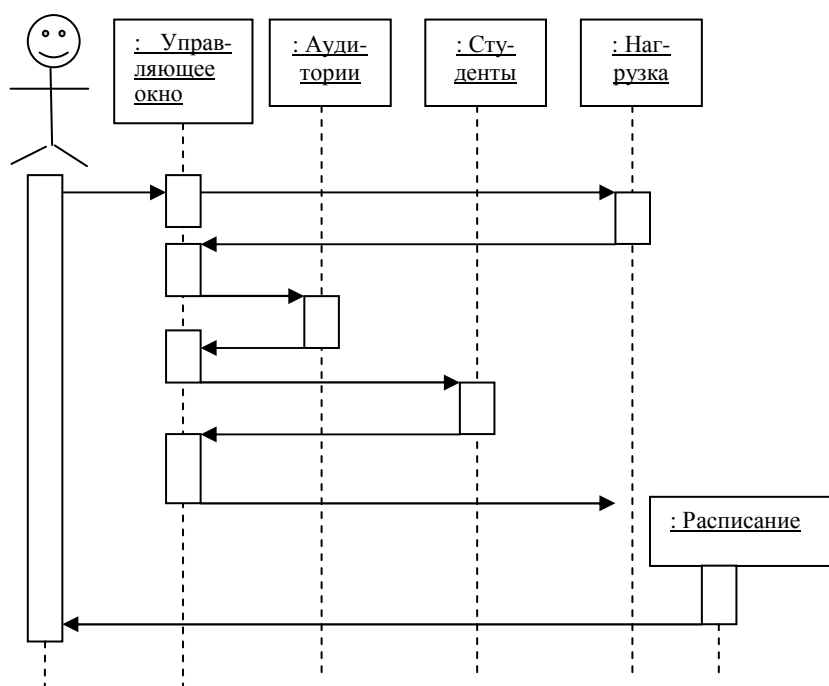


Рисунок 116 – Диаграмма последовательности

На диаграмме видно, что инициатором взаимодействия является диспетчер-составитель, который обладает фокусом управления на протяжении всей работы системы, постоянно контролируя ее деятельность. С объектов «Нагрузка», «Аудиторный фонд» и «Студенты» собирается соответствующая информация путем перехода фокуса управления от управляющего окна к соответствующему объекту и обратно. Когда собрана вся необходимая информация, создается объект «Расписание», получающий временный фокус управления.

Диаграмма состояний описывает процесс изменения состояний системы или ее подсистемы при реализации всех вариантов использования. При этом изменение состояний отдельных элементов системы может быть вызвано внешними воздействиями со стороны других элементов или извне системы. Именно для описания реакции системы на подобные внешние воздействия и используются диаграммы состояний (рис. 117).

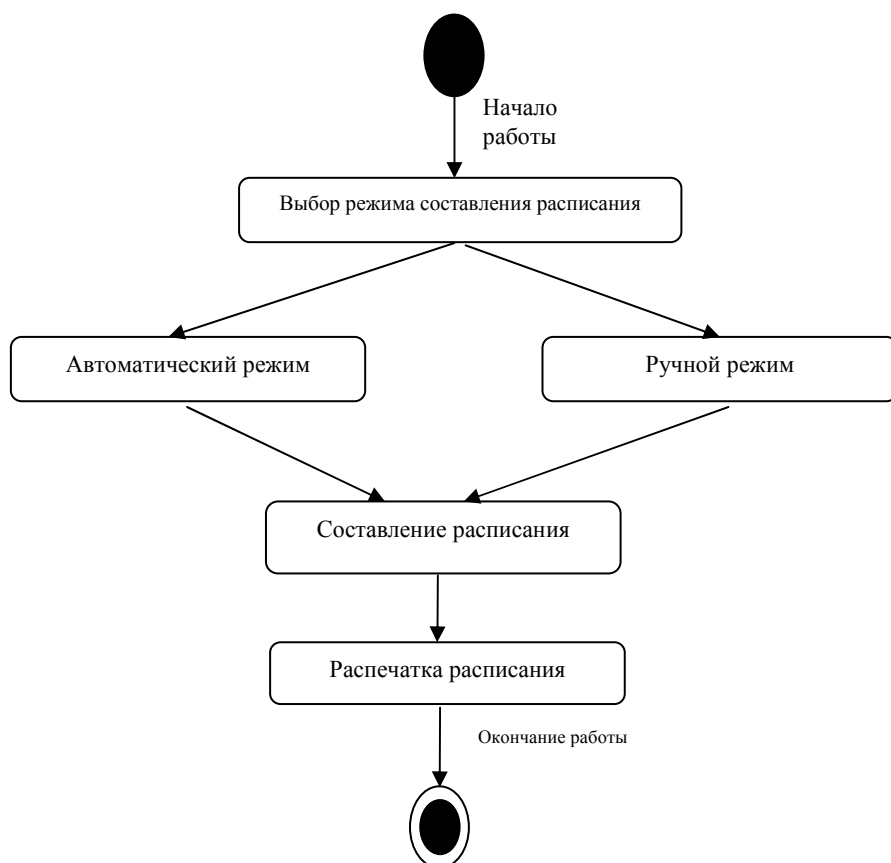


Рисунок 117 – Диаграмма состояний

Для моделирования процесса выполнения операций в языке UML используются так называемые диаграммы деятельности. Каждое состояние на этой диаграмме соответствует выполнению некоторой элементарной операции, а переход в следующее состояние срабатывает только при завершении операции в предыдущем состоянии. Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия, а дугами – переходы от одного состояния действия к другому. Таким образом, диаграммы деятельности можно считать частным случаем диаграмм состояний (рис. 118).

Для создания конкретной физической системы необходимо некоторым образом реализовать все элементы логического представления в конкретные материальные сущности. Для описания таких реальных сущностей предназначен другой аспект модельного представления, а именно – физическое представление модели. В языке UML для физического представления моделей систем используются так называемые диаграммы

компонентов и диаграммы развертывания, которые позволяют определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами (рис. 119).

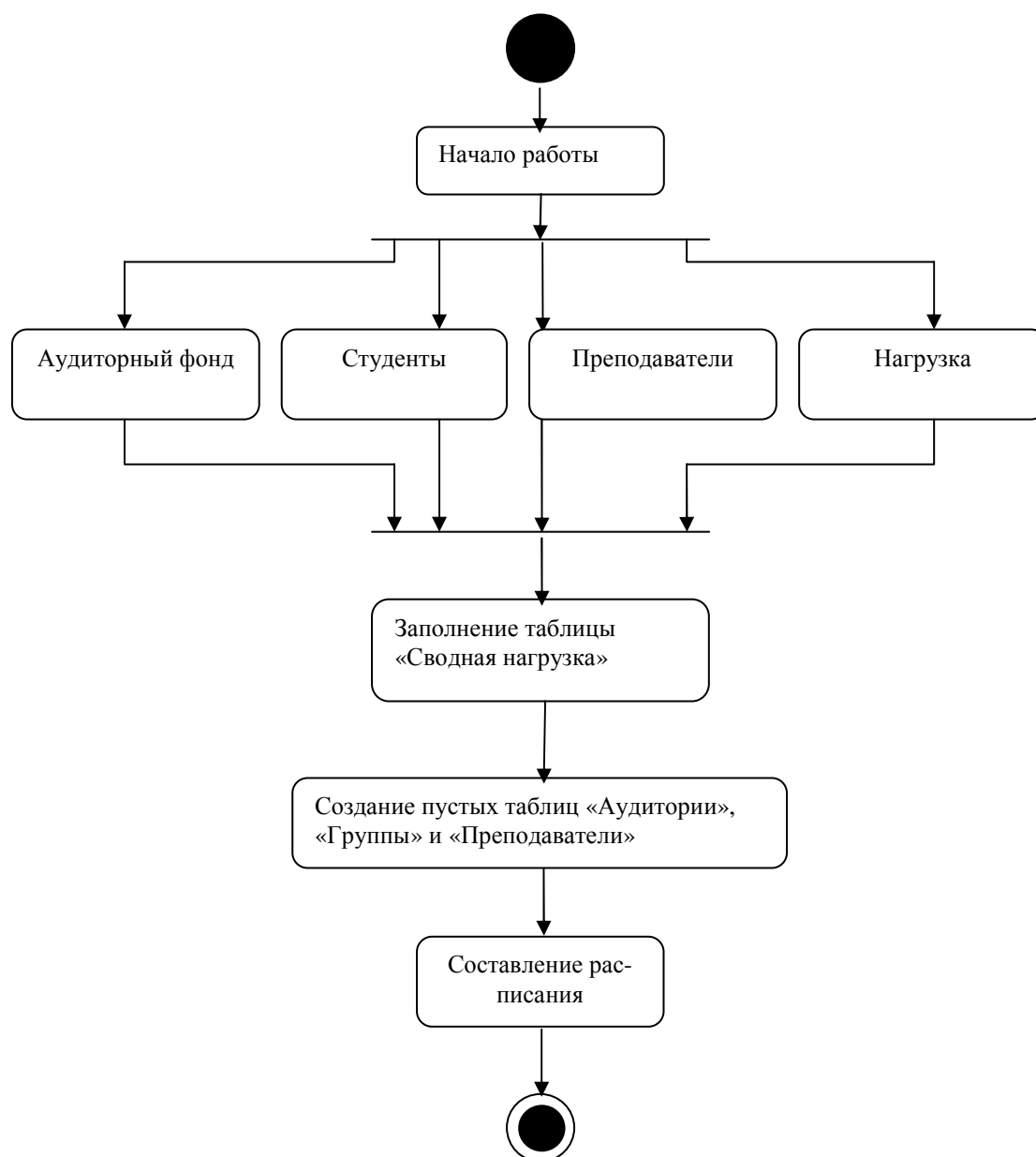


Рисунок 118 – Диаграмма деятельности

Спроектированная информационная система для автоматизации процесса составления расписания занятий была реализована в среде Borland Delphi [14-19].

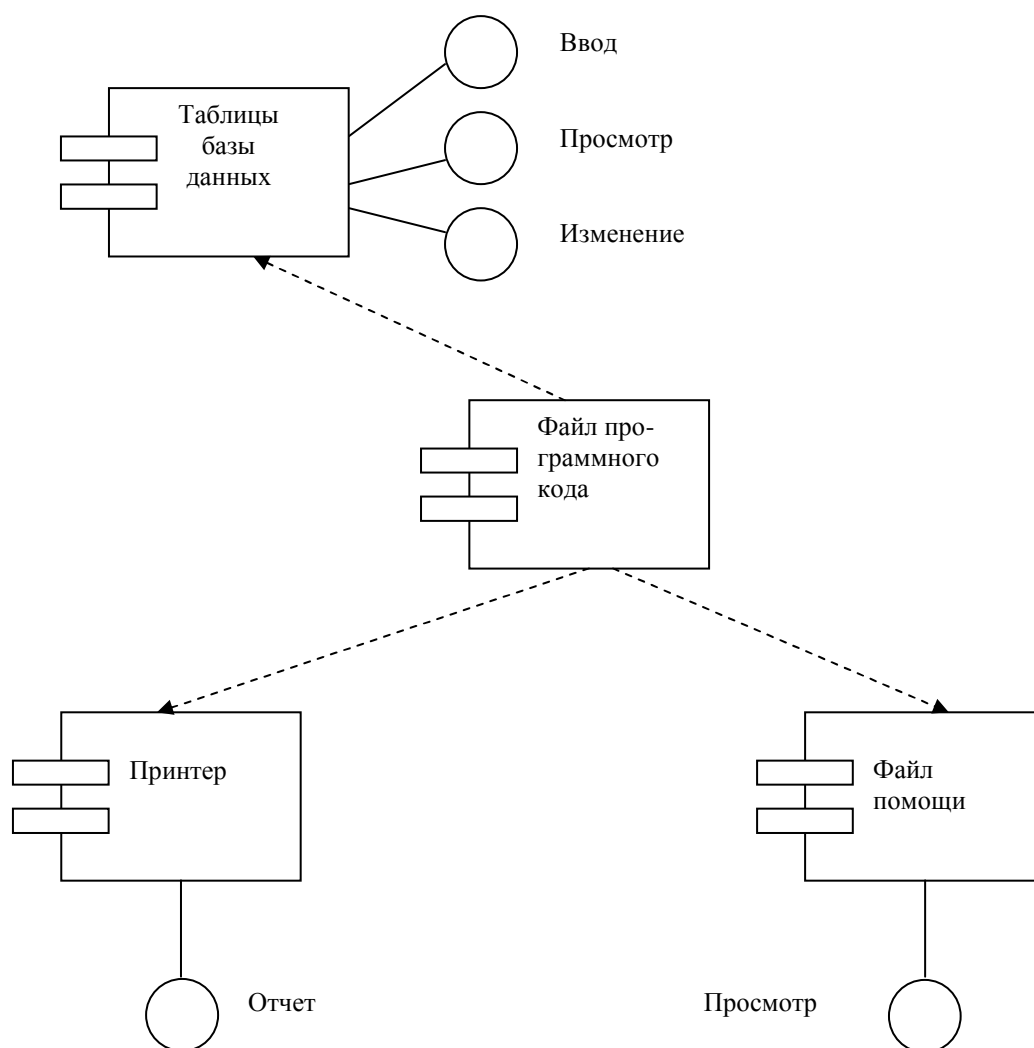


Рисунок 119 – Диаграмма компонентов

Рассчитанный коэффициент экономической эффективности системы оказался равным 3,43; срок окупаемости затрат на внедрение информационной системы составил 0,29 лет.

4.3 Информационная система для специализированного торгового предприятия

Работа специализированного торгового предприятия «Магазин-салон по продаже компакт-дисков» предполагает учет разновидностей компакт-дисков (Soft, Education, Game, Video CD, DVD, MP3, MP4, Au-

dio CD), форматов записи информации на них (Стандартный, Video CD, DVD, MP3, MP4, Audio CD), категории информации, которая может распространяться на магнитных дисках, и т.п. Кроме того, помимо собственно продажи дисков, осуществляется их прокат, запись информации на носители клиента, а также продажа дополнительных товаров (дискет, CD-R, CD-RW, полочек под диски и др.). Для успешного функционирования такого предприятия необходимо использование информационных систем и технологий: с помощью эффективной информационной системы можно значительно упростить процессы контроля и управления. Как правило, за отделом продажи компакт-дисков закреплен один компьютер, использующийся для учета наличия компакт-дисков, учета проведенных за день операций по продаже и проверке всех видов дисков на работоспособность.

В настоящее время для учета движения товара недостаточно использовать пакет MS Office – необходима информационная система или универсальная программа массового назначения, настроенная под конкретное предприятие с конкретным видом товара. Другими словами, для специализированного предприятия требуется специализированная программа учета и контроля.

Существуют уже разработанные универсальные программы массового назначения, которые упрощают учет наличия товара на складе, учет движения товара (поставки, продажа и т.д.). Примером может служить система программ «1С:Предприятие». Она состоит из нескольких модулей (компонент), которые решают конкретные четко поставленные задачи: «1С:Бухгалтерия», «1С:Зарплата и кадры», «1С:Производство» и «1С:Торговля и склад». Последний автоматизирует работу на всех этапах деятельности предприятия, содержит средства обеспечения сохранности и непротиворечивости информации, может быть адаптирован к любым особенностям учета на конкретном предприятии. Однако у пакета «1С:Торговля и склад» есть несколько недостатков:

- Достаточно высокая стоимость. Однопользовательская версия программы обойдется предприятию в \$280; 3-пользовательская сетевая версия программы – \$480; файл-серверная сетевая без ограничения

кол-ва пользователей – \$960. Дополнительно к этой стоимости ежемесячные обновления программы будут стоить от \$20 до \$100.

- Сложности в настройке программы. Для настройки программы «1С:Торговля и склад» под конкретное специализированное предприятие, а также для ее дальнейшего обслуживания необходимо либо создание дополнительного рабочего места, либо дообучение сотрудников.

Поэтому можно сделать вывод, что для учета движения товара и контроля деятельности предприятия по продаже и прокату компакт-дисков создание новой информационной системы со всеми необходимыми функциональными возможностями – наиболее дешевый и наименее трудоемкий путь.

Для создания информационной модели проектируемой системы используется унифицированный язык моделирования UML. Сначала формулируются требования к разрабатываемой системе, определяются функции, которые она должна реализовать, и задачи, которые она должна решать. На основе этих данных формируется диаграмма вариантов использования (рис. 120).

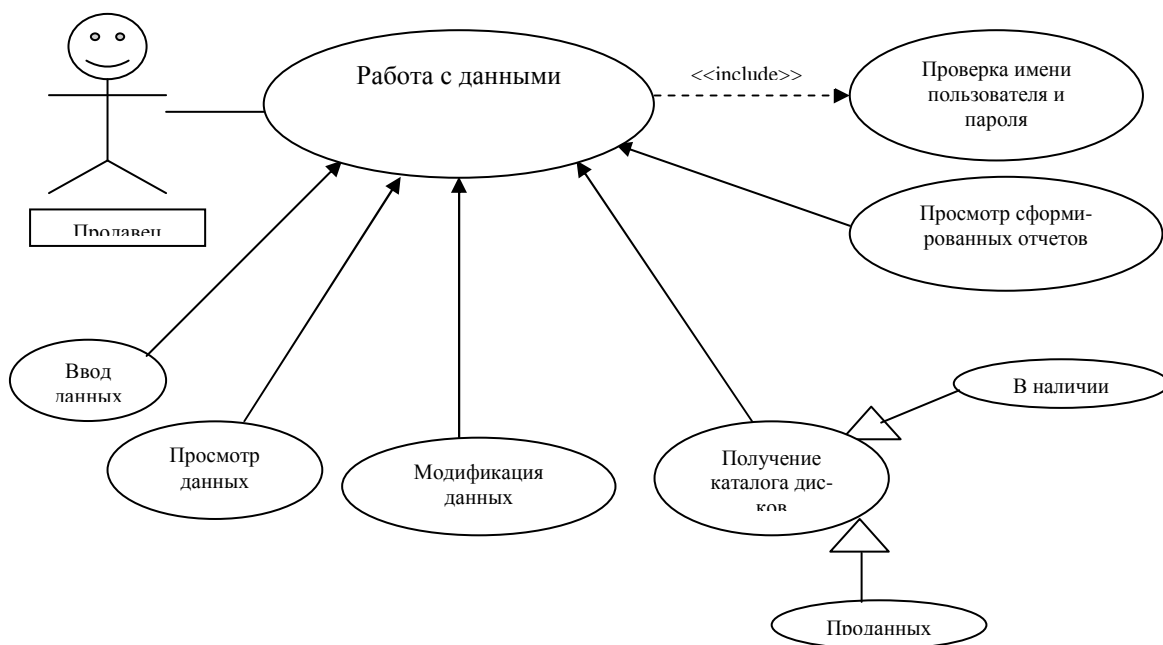


Рисунок 120 – Диаграмма вариантов использования

Диаграмма вариантов использования описывает функциональное назначение системы или, другими словами, то, что система будет делать в процессе своего функционирования. Диаграмма вариантов использования является исходным концептуальным представлением или концептуальной моделью системы в процессе ее проектирования и разработки.

На диаграмме видно, что в процессе работы с программой будет работать один актер – продавец-консультант. Предполагается наличие таких вариантов использования системы, как: ввод данных, просмотр данных, модификация данных, получение каталога дисков, просмотр сформированных отчетов.

Диаграмма классов служит для представления статической структуры модели системы, различных взаимосвязей между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описания их внутренней структуры и типов отношений. Диаграмма классов нашей информационной системы представлена на рис. 121. Здесь изображены следующие классы: интерфейс, таблица, постоянные клиенты, касса, поставщики, история и управляющее окно. Класс Таблица является абстрактным, который не имеет экземпляров и объектов. Все операции данного класса характеризуются областью видимости типа public, т.е. эти операции видны и доступны из любого другого класса. Стереотип «control» класса Управляющее окно указывает на то, что это управляющий класс, отвечающий за координацию действий других классов. Не закрашенный треугольник, направленный к нему от остальных классов, указывает на наличие отношения ненаправленной бинарной ассоциации, т.е. наличие произвольного отношения или взаимосвязи между классами. Так, классы Таблица, Интерфейс, Постоянные клиенты и Поставщики передают данные в класс Управляющее окно, а класс Управляющее окно позволяет представить данные в виде отчетов по истории проведенных операций и по динамике величин касс за определенное время.

Для рассмотрения взаимодействия объектов в контексте статической структуры модели для представления структурных особенностей передачи и приема сообщений между объектами используется диаграмма

кооперации. Данная диаграмма визуализирует объекты (экземпляры классов), связи (экземпляры ассоциаций) и сообщения.

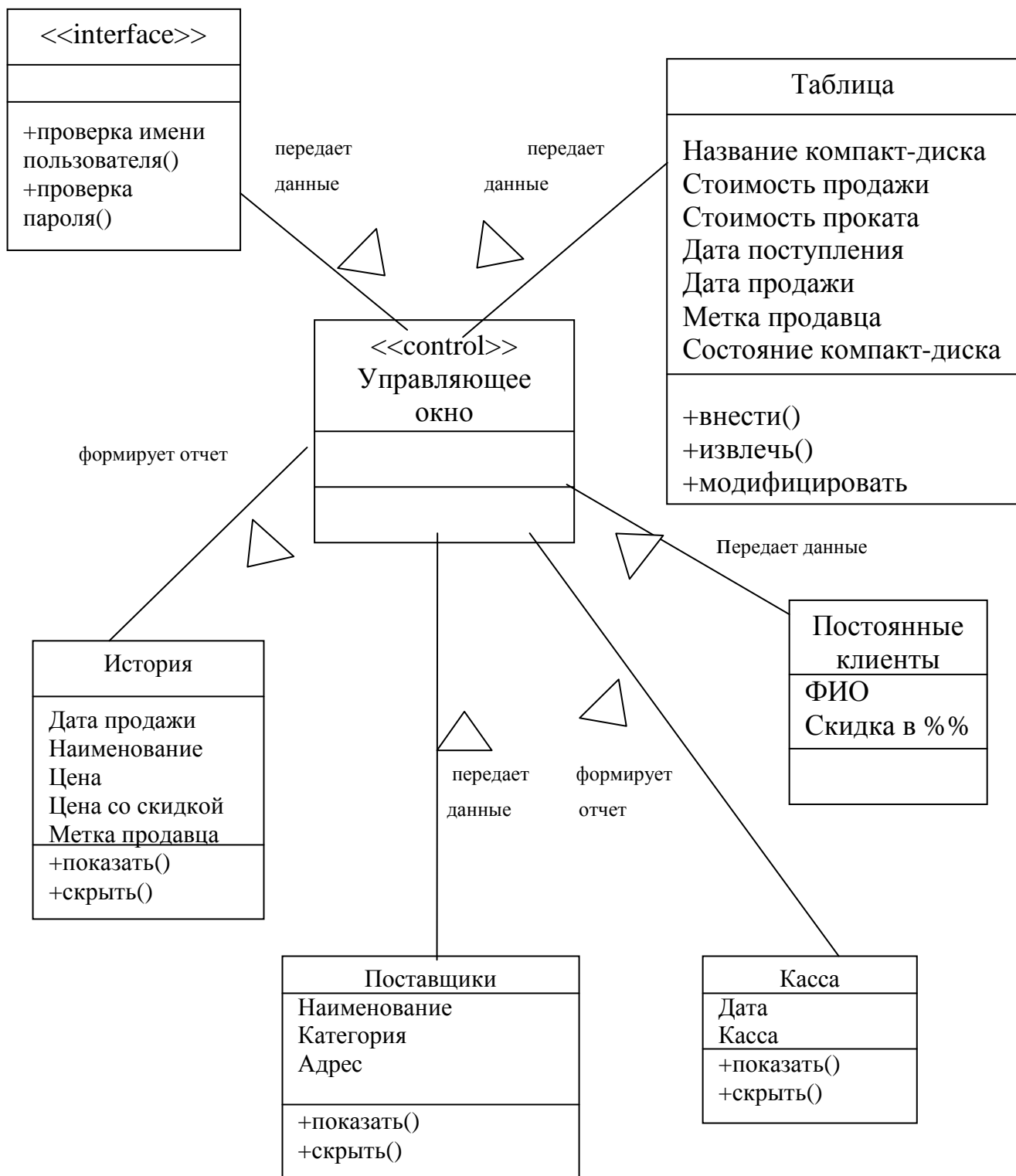


Рисунок 121 – Диаграмма классов

Диаграмма коопераций нашей информационной системы приведена на рис. 122.

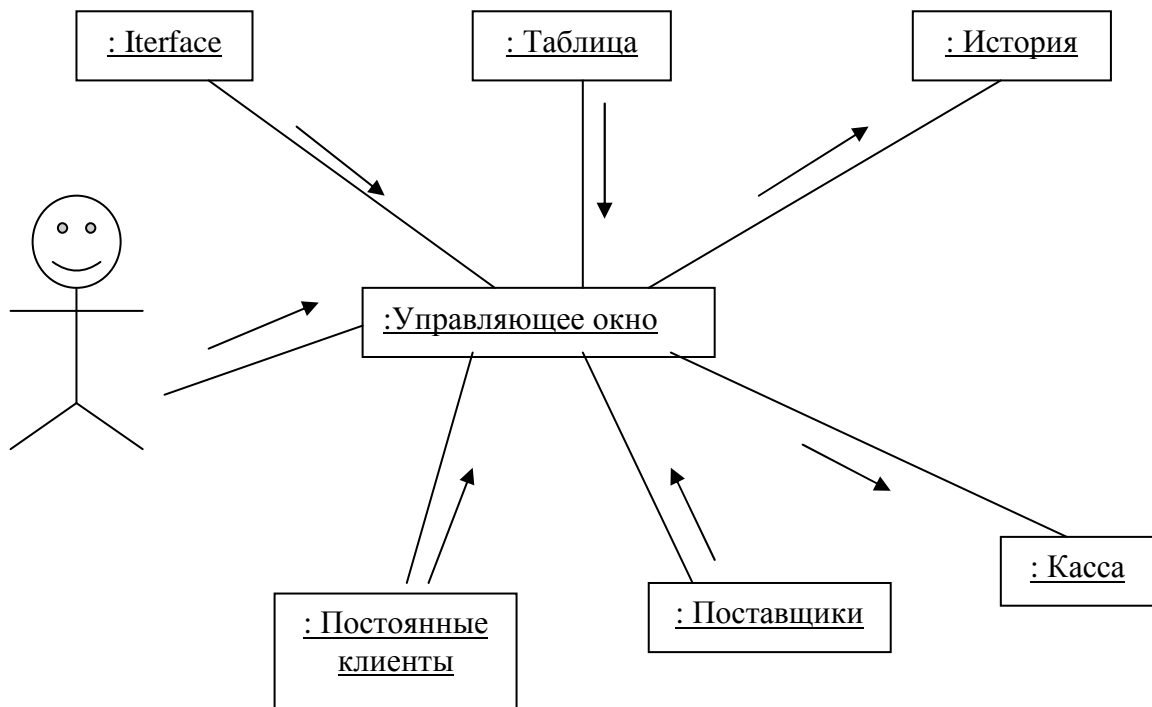


Рисунок 122 – Диаграмма коопераций

Здесь представлены такие классы: таблица, интерфейс, история, касса, поставщики, постоянные клиенты. Работа в системе имеет следующий порядок: продавец-консультант начинает работу с управляющим окном, введя своё имя и пароль. Далее он инициирует передачу данных о поставщиках из класса «Поставщики», постоянных клиентах из класса «Постоянные клиенты», формирует отчеты «Касса» и «История», работает непосредственно с данными таблицы.

Диаграмма последовательности отражает временной аспект поведения системы (рис. 123). На диаграмме видно, что инициатором взаимодействия является продавец-консультант, который обладает фокусом управления на протяжении всей работы системы, постоянно контролируя ее деятельность.

Для моделирования процесса выполнения операций в языке UML используются диаграммы деятельности. Деятельность представляет собой некоторую совокупность отдельных вычислений, выполняемых ав-

томатом; при этом отдельные элементарные вычисления могут приводить к некоторому результату или действию.

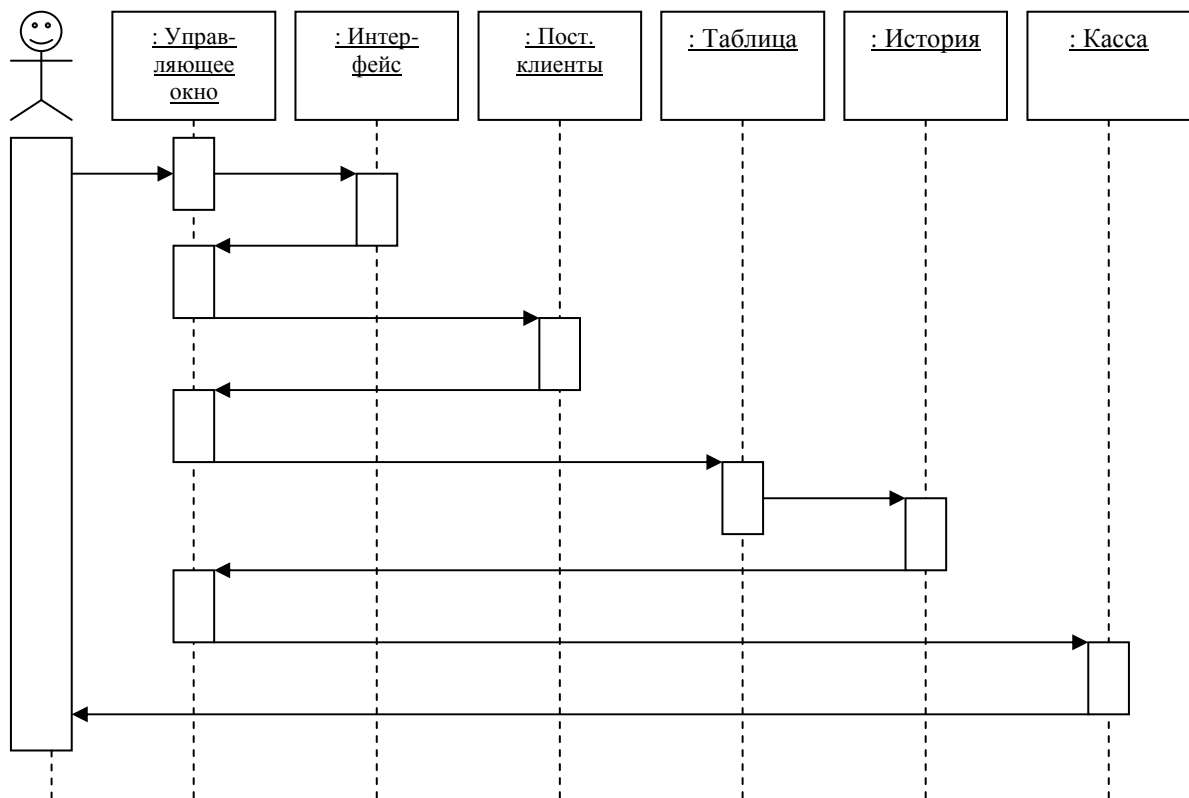


Рисунок 123 – Диаграмма последовательности

Диаграмма деятельности информационной системы изображена на рис. 124. По ней видно, что на после обращения клиента проверяется наличие диска (визуально либо с помощью информационной системы). Если диск есть в наличии, и он удовлетворяет всем требованиям, проводится проверка, клиент постоянный или нет, и вид операции (продажа или прокат). В зависимости от полученной информации диск списывается в продажу или прокат по обычной цене или по цене со скидкой.

Для создания конкретной физической системы необходимо некоторым образом реализовать все элементы логического представления в конкретные материальные сущности. Для описания таких реальных сущностей предназначен другой аспект модельного представления, а именно – физическое представление модели. В языке UML для физического представления моделей систем используются так называемые диаграммы компонентов, которые позволяют определить архитектуру разрабаты-

ваемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код.

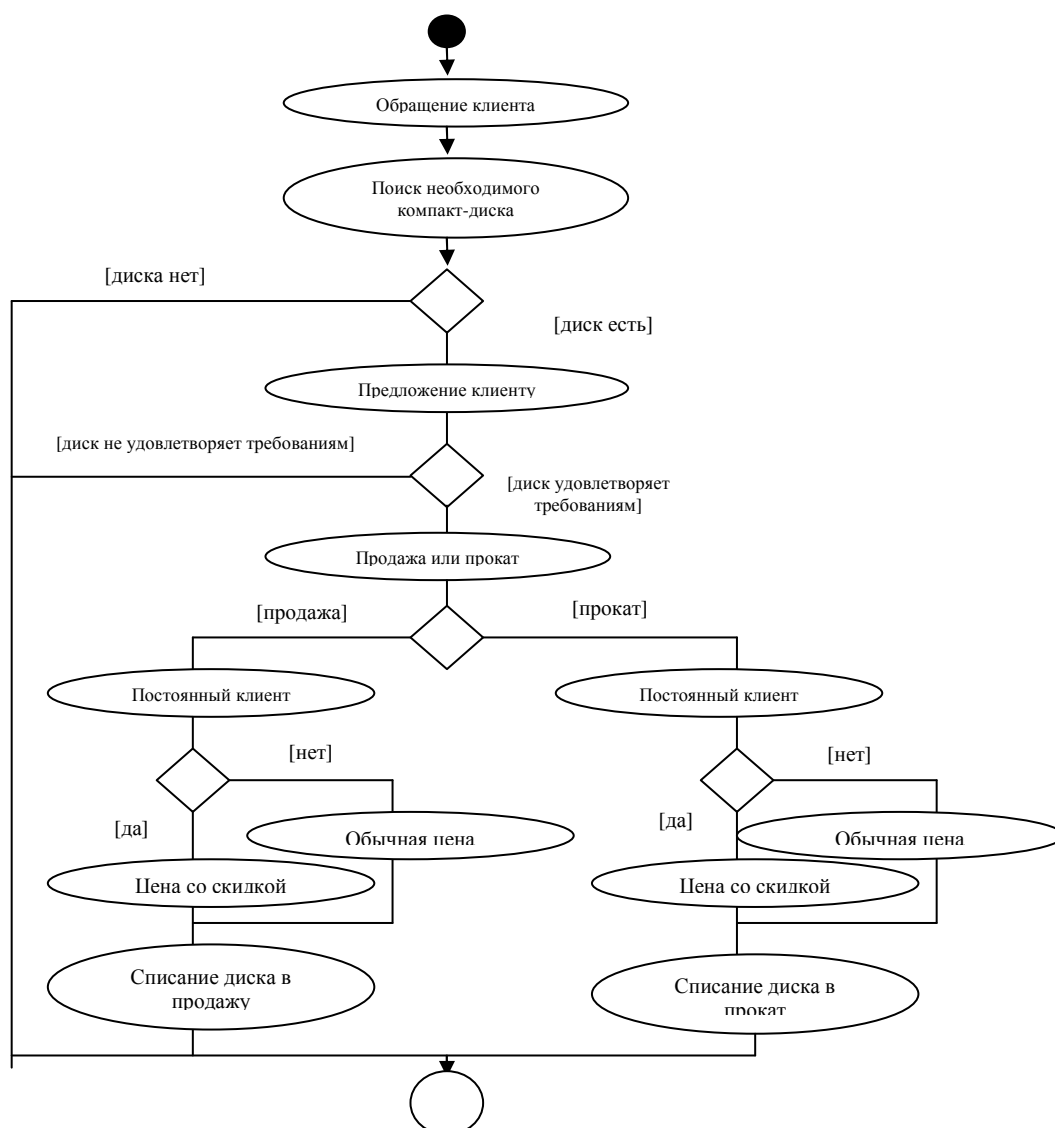


Рисунок 124 – Диаграмма деятельности

Во многих средах разработки модуль или компонент соответствует файлу. Диаграмма компонентов обеспечивает согласованный переход от логического представления к конкретной реализации проекта в форме программного кода.

Система была реализована в среде программирования Borland Delphi и внедрена на торговом предприятии [20-21]. Проведенный расчет экономической эффективности показал, что внедрение информационной

системы позволит сэкономить 1035,5 грн. в год за счет высвобождения рабочего времени работников отдела по продаже компакт-дисков. При этом будет увеличена скорость обслуживания клиентов, что отразится на величине выручки и, соответственно, и на прибыли.

4.4 Информационная система для небольшой страховой компании

В страховом бизнесе, как и в любой другой области экономики, идет непрерывный поиск эффективных технологий и методов повышения рентабельности. Современное рыночное общество невозможно себе представить без страхования как особого вида экономических отношений. Существует прямая связь между уровнем благосостояния общества, степенью развития рыночных отношений и уровнем развития страхования. В странах, которые являются мировыми лидерами в области социальных и рыночных отношений (США, Япония, европейские государства и другие), страхование есть одной из наиболее стабильных и динамических областей народного хозяйства.

Развитие страховой деятельности на Украине сопровождается рядом проблем. Одной из наиболее важных проблем страхования есть медленная (в сравнении с банковской деятельностью) автоматизация страхования. Эта проблема наиболее остро проявляется в получении оперативной информации относительно заключенных договоров страхования, что, в свою очередь, не разрешает в полной мере реагировать на внешние воздействия и эффективно принимать управленческие решения. Особенно это касается небольших страховых компаний, которые мало известны на страховом рынке.

На данное время наиболее популярными являются разные решения, разработанные на основе современных информационных технологий. Для страховых компаний рынок современных IT-технологий предлагает разные по аппаратному составу и программному обеспечению – от отдельных систем, наборов приложений и модулей к универсальным системам, которые обеспечивают довольно полную автоматизацию всех производственных, технологических и вспомогательных процессов.

Так, модульная информационная система INSTRAS-4 предназначена для комплексной автоматизации учета в страховых и перестраховочных компаниях. Основное назначение INSTRAS-4 – повышение конкурентоспособности страховой компании за счет важного улучшения управляемости. Замыкает все данные и все бизнес-процессы в страховой компании в единое информационное пространство, согласовывая и оптимизируя их. Обеспечивает комплексную автоматизацию всех ключевых отделов и служб страховой (перестраховочной) компании. Представляет собой могущественный инструмент преобразований и совершенствования работы компании, создания и вывода на рынок новых страховых продуктов. Разрешает организовать бесперебойный обмен информацией с филиалами, агентствами и точками продаж. Включает интерфейс к наиболее популярным на украинском рынке программам бухгалтерского учета. Допускает минимальные расходы на приобретение, внедрение, поддержку, сопровождение и развитие.

Для использования в страховом маркетинге предлагается еще целый ряд программных продуктов. Например, Marketing Expert и БЕСТ Маркетинг – для стратегического планирования, Касатка и Маркетинг Микс – для стратегического и оперативного планирования. Программа Marketing Analytic предназначена для анализа, прогноза, планирования и содержит в себе элементы CRM. Программа Sales Expert обеспечивает управление прямыми продажами. Специализированные программные продукты, предназначенные для решения задач планирования, имеют и другие встроенные маркетинговые инструменты, используемые для SWOT-анализа и ряда других маркетинговых операций в процессе обработки данных. В той или иной мере вышеперечисленные программные продукты можно применять с целью прогнозирования и разработки сценариев развития событий в интересах решения маркетинговых задач страховой компании.

Рынок IT-технологий предлагает и другие информационные решения. Например, система Contact Manager является универсальным средством администрирования рабочего времени и управления системой сбыта. Она позволяет эффективно контролировать деловую активность сотрудников системы продаж на всех уровнях руководства, отслеживать

историю контактов специалистов агентской сети – от первого телефонного звонка к заключению договора. Contact Manager интегрирован с системой операционного учета договоров. Возможна интеграция с технологиями call-центра, бизнес-планирования и отчетности.

Новая версия информационной системы UNICUS EASY ОСАГО v. 1.6. представляет собой автоматизированное рабочее место (АРМ) работника страховой компании и позволяет выполнять следующие операции: полнофункциональную продажу полисов ОСАГО; урегулирование убытков; учет бланков строгой отчетности; экспорт данных в формате XML во внешние информационные базы.

Система ICI (Insurance Company Information System), которая выпускается фирмой T-systems, адаптируется к разнообразнейшим бизнесам-моделям и может рассматриваться как промежуточный вариант между индивидуальным и стандартным программным обеспечением. В этой системе комплексной автоматизации, разработанной специально для страховых компаний, есть такие современные функции, как CRM, управление документооборотом, ведение статистики и другие. Эта система легко интегрируется с уже имеющимися в страховой компании бэк-офисными решениями. Программа Connect Insurance, разработанная австрийской компанией, работает по принципу “time to market” (процесс от возникновения идеи о новом страховом продукте до его выхода на рынок) и модернизации системы продаж страховой компании без внесения изменений в существующую IT архитектуру.

Но надо помнить, что интегрированные информационные системы типа Insurance Company используются, в основном, большими страховыми компаниями, так как имеют высокую рыночную стоимость и потому недоступны для мелких страховых компаний. Для автоматизации своей деятельности мелкие страховые компании используют или системы собственных разработок, или недорогие системы, которые предлагают ограниченный набор возможностей.

Мы предлагаем информационную систему, рассчитанную на большой круг пользователей, т.е. для тех офисных рабочих, работа которых непосредственно связана с базой данной информационной системы. Проектируемая система не исключает использования ее любыми не-

большими страховыми компаниями, предполагает полную автоматизацию деятельности страховой компании и поддерживает полный цикл процесса обработки информации по всем видам страхования.

Разработка информационной системы традиционно проводится в три этапа: объектно-ориентированный анализ предметной области; концептуальное, логическое и физическое моделирование информационной системы; программная реализация разработанной модели.

Первый этап включает анализ действий, осуществляемых в процессе страхования, выделение активных и пассивных классов (объектов). Актером (активным объектом) в нашем случае будет оператор ввода данных (в первом приближении), а пассивными объектами – данные по заключенным договорам страхования.

На втором этапе создается информационная модель проектируемой системы, для чего используется унифицированный язык моделирования UML.

Проектирование начинается с формулирования требований к разрабатываемой системе, определения функций, которые она должна реализовывать, и задач, которые она должна решать. На основе этих данных формируется диаграмма вариантов использования (рис.125). В нашем случае диаграмма вариантов использования («прецедентов») основывается на требованиях, функциях, задачах страховой компании, которые должна выполнять разрабатываемая система. Согласно диаграмме, прецедент «Работа с данными» является основой для прецедентов: получение данных; введение данных; модификация данных; просмотр данных; контроль правильности введенных данных.

Прецедент «Обработка данных» является основой для прецедентов: просмотр данных; сверка данных; формирование отчета; выдача исходной информации.

Прецедент «Работа с исходной информацией» является основой для таких прецедентов: просмотр данных; обработка исходной информации; формирование отчетов.

Дальше строится структурная схема системы в виде диаграммы классов. Диаграммы классов представляют собой отправную точку процесса разработки. Диаграммы классов помогают при анализе предметной

области. Они позволяют аналитику общаться с клиентом в его терминологии и стимулируют процесс выявления важных деталей в проблеме, которую нужно решить.

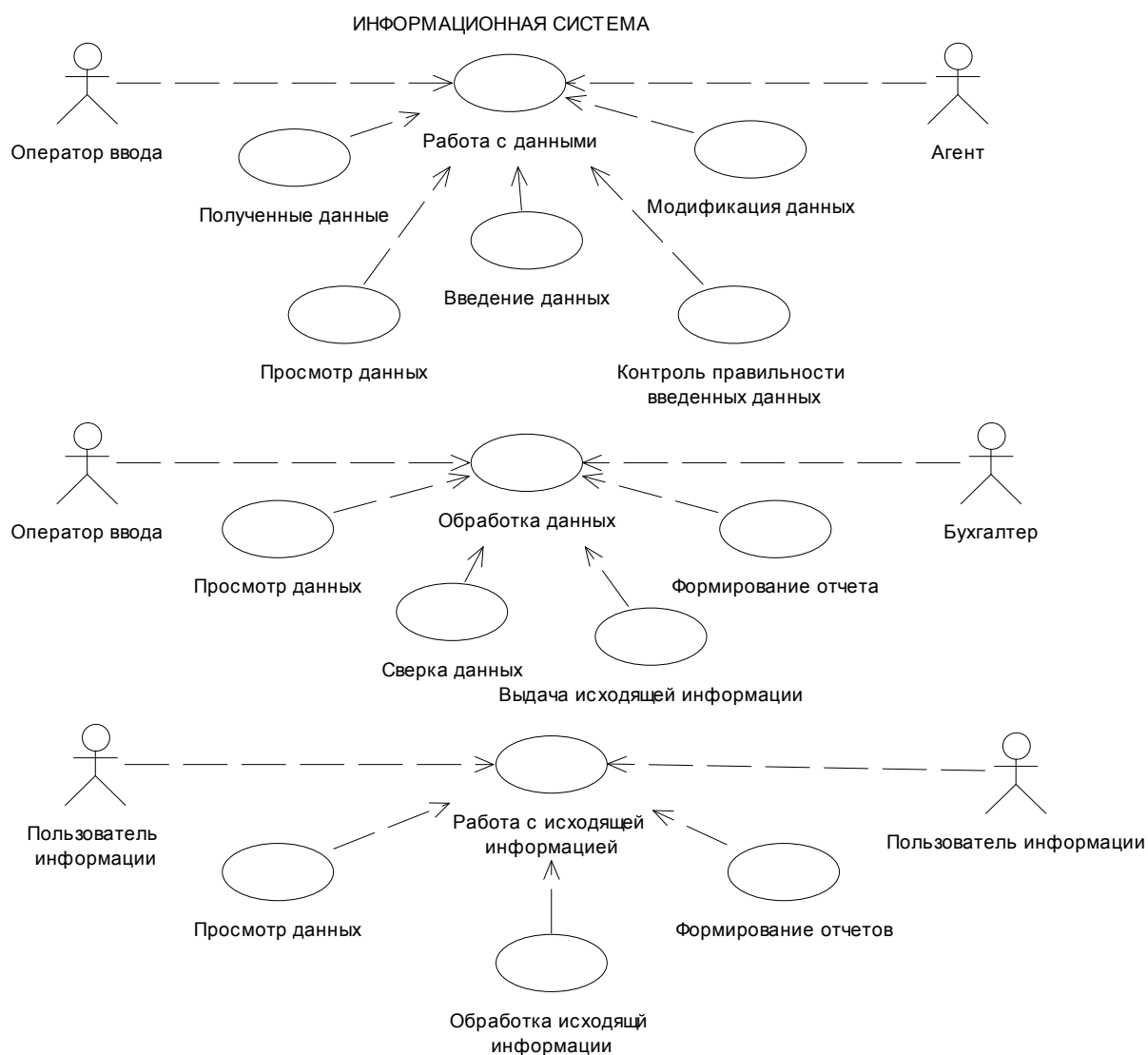


Рисунок 125 – Диаграмма вариантов использования

Анализ предметной области процесса функционирования небольшой страховой компании изображен на рис. 126.

Клиент-страхователь обращается в страховую компанию к клиенту-агенту с целью заключить договор страхования. Согласовываются условия страхования, подписывается договор страхования.

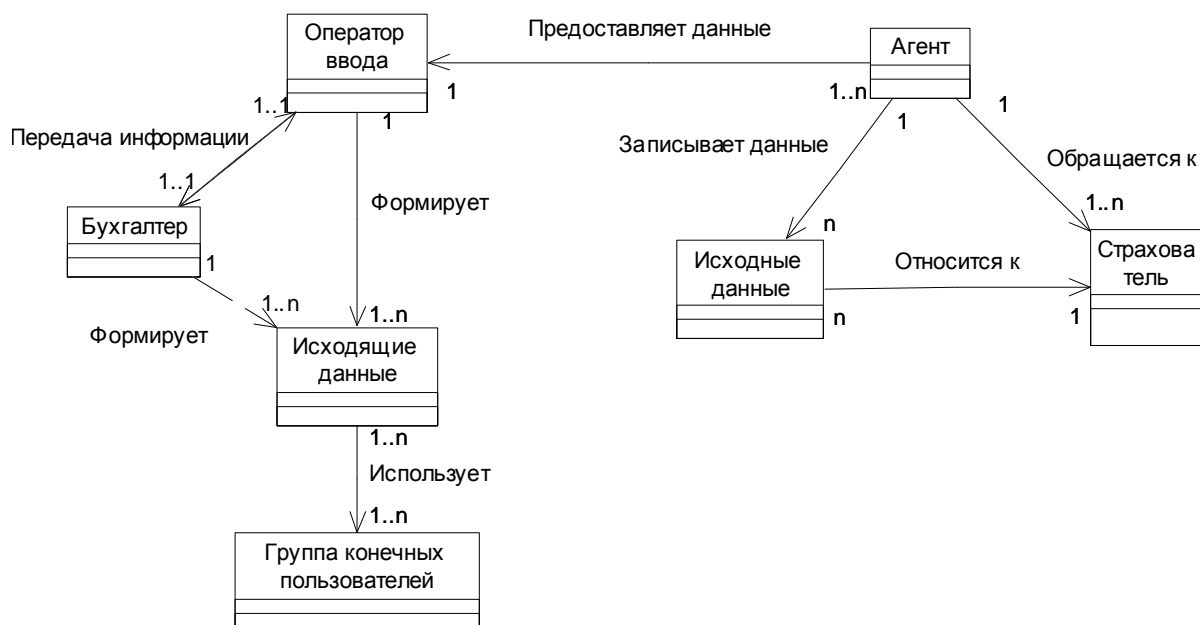


Рисунок 126 - Диаграмма ассоциаций классов

Клиент-агент передает записанные исходные данные клиента-страхователя (страховой полис) клиенту-оператору ввода, который, в свою очередь, обрабатывает эти данные и заносит в ПК (сохранение в базу). Также клиенту-оператору ввода поступают бухгалтерские данные от клиента-бухгалтера. После того, как все данные обработаны и сохранены в базе, клиент-оператор ввода формирует исходную информацию (в виде отчета определенной формы) и передает клиенту-бухгалтеру для сверки. В свою очередь, клиент-бухгалтер после просмотра и обработки полученной информации также формирует исходную информацию (в виде отчета определенной формы). Сформированные и обработанные данные, как клиентом-оператором ввода, так и клиентом-бухгалтером передаются клиенту-группе конечных пользователей. Вербальное описание процесса работы страховой компании с клиентами позволяет выделить следующие классы: клиент-страхователь, клиент-агент, клиент-оператор ввода, клиент-бухгалтер, клиент-группа конечных пользователей, входящие данные, исходящие данные. На рис. 127 представлена детализация классов.

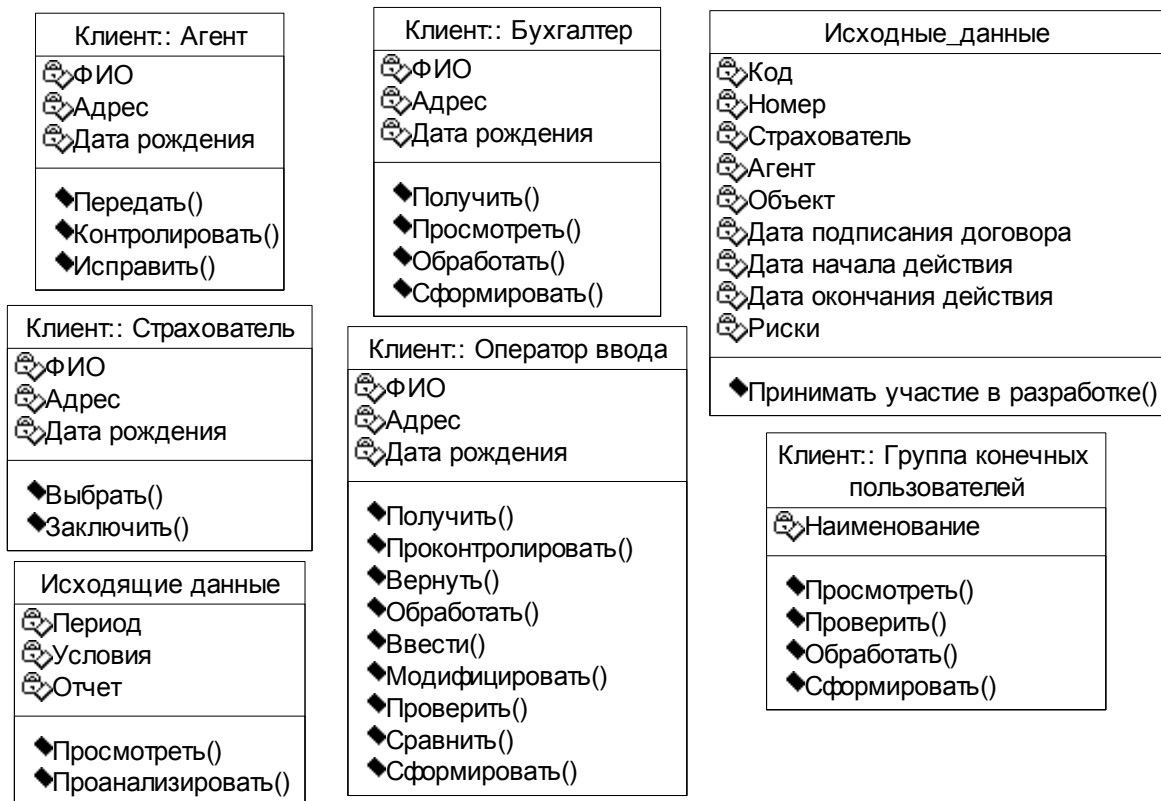


Рисунок 127 – Диаграмма детализации классов

Диаграмма последовательностей показывает измерение времени по взаимодействию объектов (рис. 128). В нашем случае фигурка представляет страхователя (исполнителя), который инициирует последовательность, хотя, в принципе, эта фигурка не является частью диаграммы последовательностей. Объект «Оператор ввода» получает сообщение от агента, после чего происходит обработка полученных данных. В процессе работы объект «Оператор ввода» получает асинхронное сообщение от объекта «Бухгалтер» (т.е. объект «Бухгалтер» не ожидает ответа от объекта «Оператор ввода»). После обработки данных объект «Оператор ввода» передает данные объекту «Бухгалтер» в виде синхронного сообщения, т.е. ожидает ответ от объекта «Бухгалтер», а именно идет уточнение со стороны объекта «Бухгалтер» в виде условия: обработанные данные верны или обработанные данные неверны. В случае данные неверны - идет доработка, а в случае данные верны - происходит отправление сообщения конечному пользователю, т.е. группе конечных пользователей.

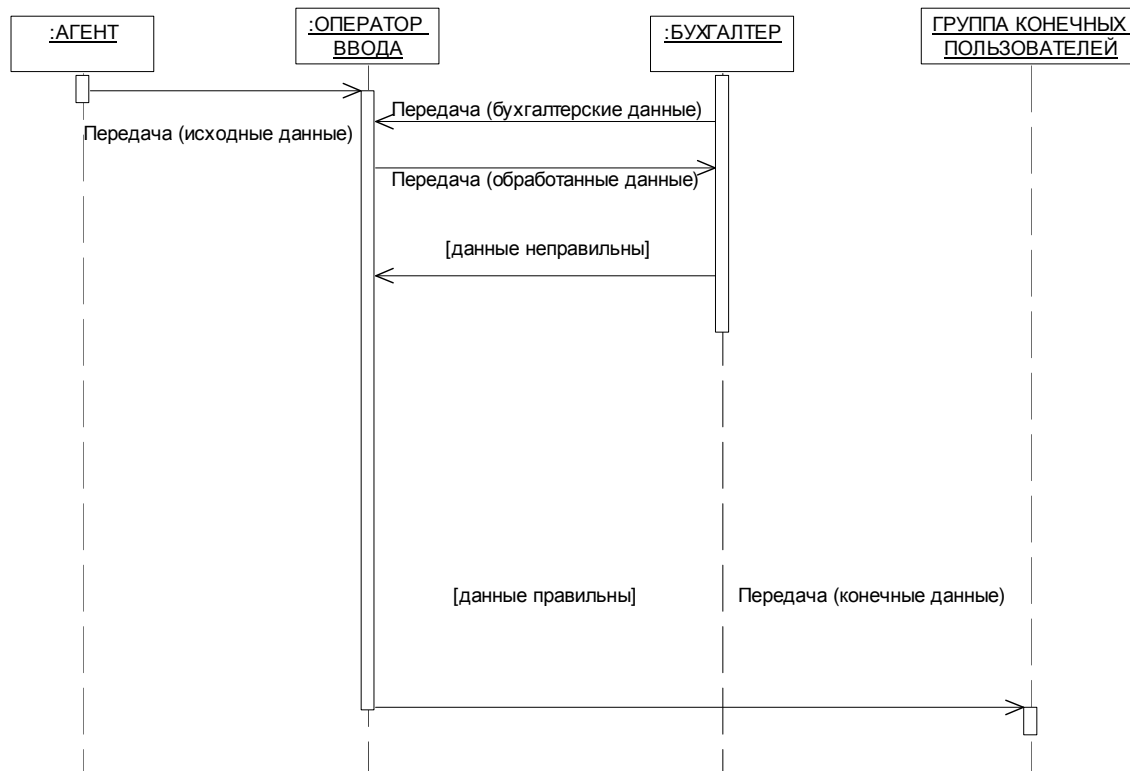


Рисунок 128 – Диаграмма последовательности

Диаграмма кооперации, так же, как и диаграмма последовательностей, отражает взаимодействие объектов. Но расхождение этих двух диаграмм заключается в том, что диаграмма последовательностей упорядочена в соответствии со временем, а диаграмма кооперации – в соответствии с пространственным расположением объектов, т.е. ориентирована на состояние и общую организацию взаимодействующих объектов.

На рис. 129 показана диаграмма кооперации работы страховой компании. Диаграмма описывает последовательность информации, которая поступает объекту в виде сообщения. В нашем случае некоторые сообщения являются дочерними относительно других. Они представлены с использованием десятичной точки для обозначения уровня вложенности. Так, объект «Оператор ввода» может передать обработанные данные объекту «Бухгалтер» только после того, как получит бухгалтерские данные от этого же объекта. Так как без бухгалтерских данных невозможно сформировать конечные данные. В свою очередь, конечные данные объект «Группа конечных пользователей» от объекта «Оператор ввода» получает только после передачи исходящих данных от объекта «Бухгалтер».



Рисунок 129 – Диаграмма кооперации

Диаграмма состояний отображает, как объекты изменяют свое состояние в ответ на события, которые происходят. При включении компьютера происходит загрузка. Поэтому включение компьютера является переключающим событием, которое приводит к переходу интерфейса в состояние инициализация, а загрузка – действие, которое происходит во время перехода. Результатом выполнения действий в состоянии инициализации есть изготовление переключающего события, которое вызывает переход в состояние работы. При щелчке по кнопке завершения работы осуществляется переход в состояние завершения работы, и в конечном итоге компьютер выключается. Состояние регистрации нового страхового полиса в базе является подчиненным. Система находится в состоянии «Ожидание ввода данных», пока оператор ввода не введет соответствующее условие. Далее активизируются состояния «Обработка данных» и «Сохранение данных». Итоговое состояние – «Визуализация результатов».

Диаграмма деятельности реально описывает все, что происходит во время операции или в процессе. В нашем случае диаграмма видов деятельности разбивается на «плавательные дорожки» – параллельные сегменты, которые отвечают ролям (рис. 130).

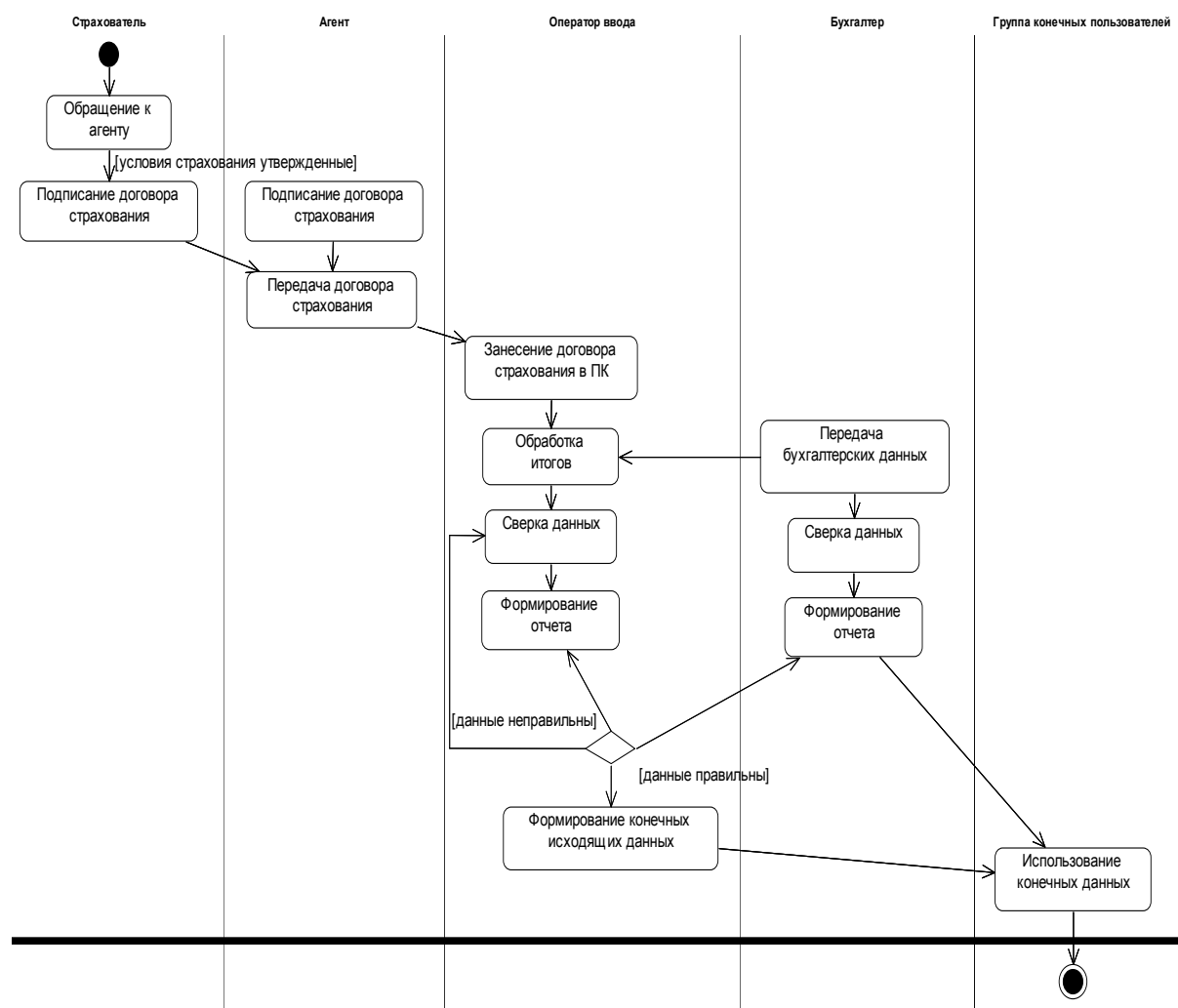


Рисунок 130 – Диаграмма деятельности

На диаграмме компонентов в нашем случае для работы с данными необходим компонент «Таблица», а для представления данных таблицы – интерфейс «Работа с данными». В свою очередь, работа с данными может содержать просмотр данных, регистрацию новых данных, редактирование данных, формирование отчета по заданному условию. Выделим следующие интерфейсы: «Просмотр», «Редактирование», «Выбор», «Автоматический подбор» и «Отчет» (рис. 131).

Система была реализована в среде программирования Borland Delphi и внедрена в отделении страховой компании [22-25]. Проведенный расчет экономической эффективности показал, что срок окупаемости системы составит 0,15 года.

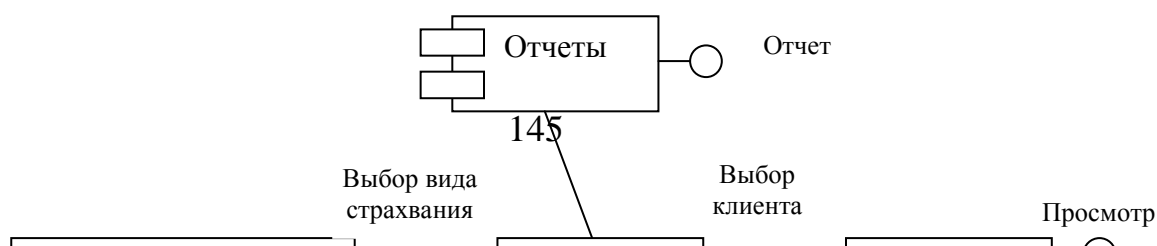


Рисунок 131 – Диаграмма компонентов

4.5 Информационная система для обеспечения функционирования финансового отдела предприятия

Во главе финансового отдела рассматриваемого предприятия стоит финансовый директор, осуществляющий управление и контроль над деятельностью службы в целом. Финансовый отдел состоит из 12 человек и подразделяется на следующие сегменты: техническая бухгалтерия, группа по налоговому учету, группа по управленческому учету, анализу и бюджетированию. Непосредственный контроль над правильностью отображения хозяйственных операций и формирования финансовой и налоговой отчетности осуществляет внутренний аудитор. Для эффективного управления затратами и формирования бюджета на предприятии формируются центры финансовой ответственности.

Финансовый отдел выполняет следующие поставленные задачи: организация финансовой деятельности предприятия, направленной на обеспечение финансовыми ресурсами заданий плана, сохранности и эффективного использования основных фондов и оборотных средств, трудовых и финансовых ресурсов предприятия, своевременности платежей по обязательствам в государственный бюджет, поставщикам и учреждениям банков.

В связи с этим идет разбиение функций финансового отдела на три области: финансово-кредитного планирования, финансово-оперативной работы и области контрольно-аналитической работы, каждая из которых возлагается на одну из трех групп отдела. Так, группа по технической бухгалтерии выполняет 19 функций из области финансово-кредитного планирования, группа по налоговому учету выполняет 13 функций в области финансово-оперативной работы, группа по управленческому учету, анализу и бюджетированию выполняет 8 функций в области контрольно-аналитической работы.

Первым этапом процесса объектно-ориентированного анализа и проектирования является построение диаграммы вариантов использования (рис. 132).

Из диаграммы следует, что пользователями системы будут три актера – это три группы, на которые делится финансовый отдел ЗАО «ГД» – техническая бухгалтерия, группа по налоговому учету и группа по управленческому учету. Рассмотрим детально варианты использования каждого актера для данной системы. В соответствии с диаграммой вариантов использования актер «Техническая бухгалтерия» отвечает за работу с первичной документацией, то есть получение данных с производства, их детальное рассмотрение, введение ежедневной бухгалтерской документации. Актер «Группа по налоговому учету» будет заниматься детальной обработкой данных: анализировать бухгалтерские данные, проверять их правильность, формировать отчеты для государственных и внутренних служб. Актер «Группа по управленческому учету» занимается исключительно выходной информацией, ее анализом, обработкой, подведением итогов и формированием выводов по деятельности отдела в целом.

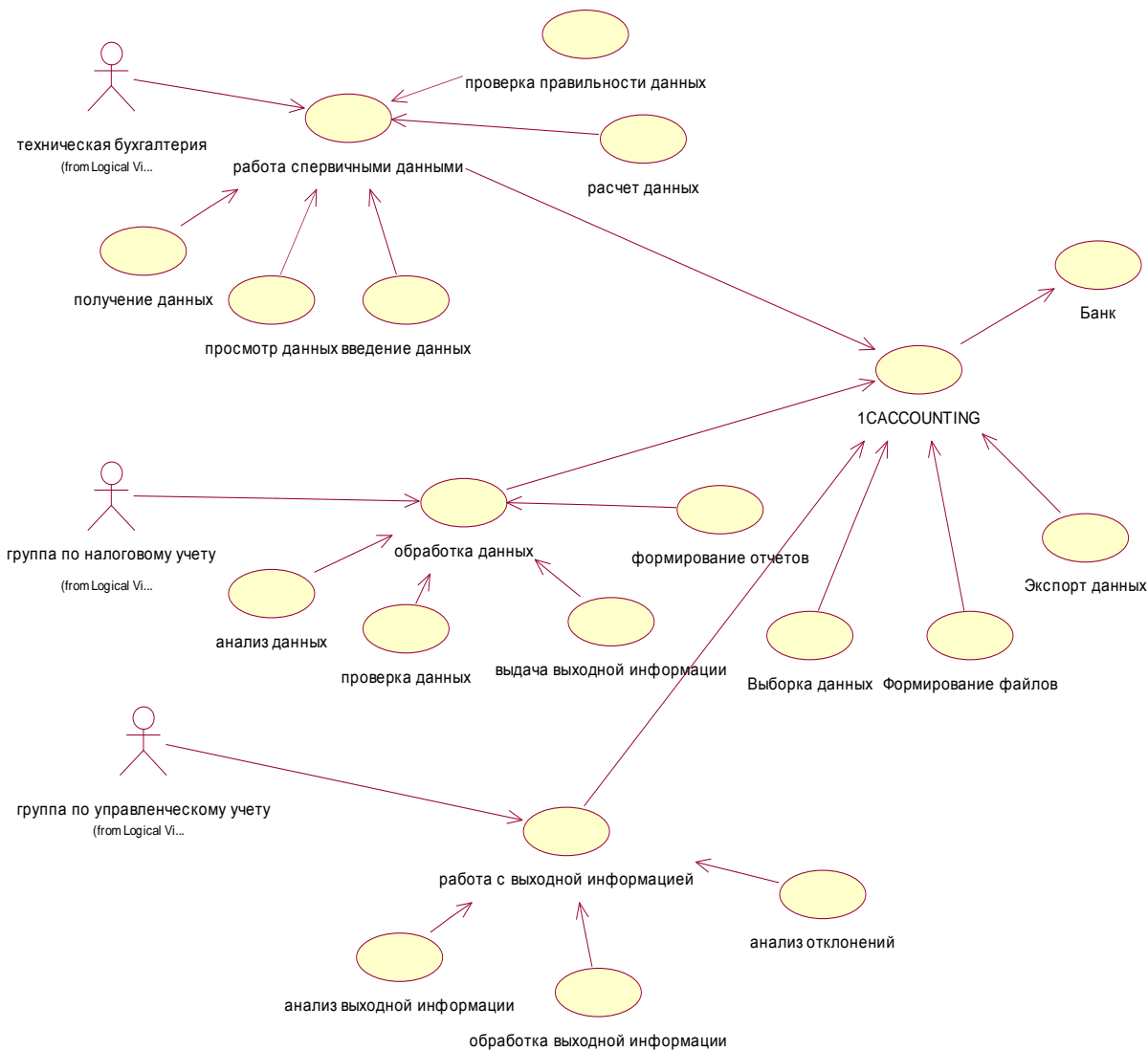


Рисунок 132 – Диаграмма вариантов использования

Для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования служит диаграмма классов (рис. 133). На ней изображены следующие классы: первичные данные, техническая бухгалтерия, группа по налоговому учету, группа по управленческому учету, выходная информация, банк, налоговая инспекция, комитет управления. Отношением зависимости указано, что классы «Техническая бухгалтерия» и «Группа по налоговому учету» зависят от класса «Первичные данные». Таким же образом класс «Выходная информация» оказывается зависимым от классов «Техническая бухгалтерия» и «Группы по налоговому учету», так как эти два класса формируют отчетность для внутренних и государственных служб учета и аудита. Далее из диаграммы следует бинарное отношение ассо-

циации от класса «Выходная информация» к классам «Банк», «Налоговая инспекция» и к «Группе по управленческому учету». Между «Группой по управленческому учету» и «Комитетом управления» устанавливается отношение зависимости, так как данная группа подотчетна финансовому директору, генеральному директору, а также акционерам, которые и составляют комитет управления.

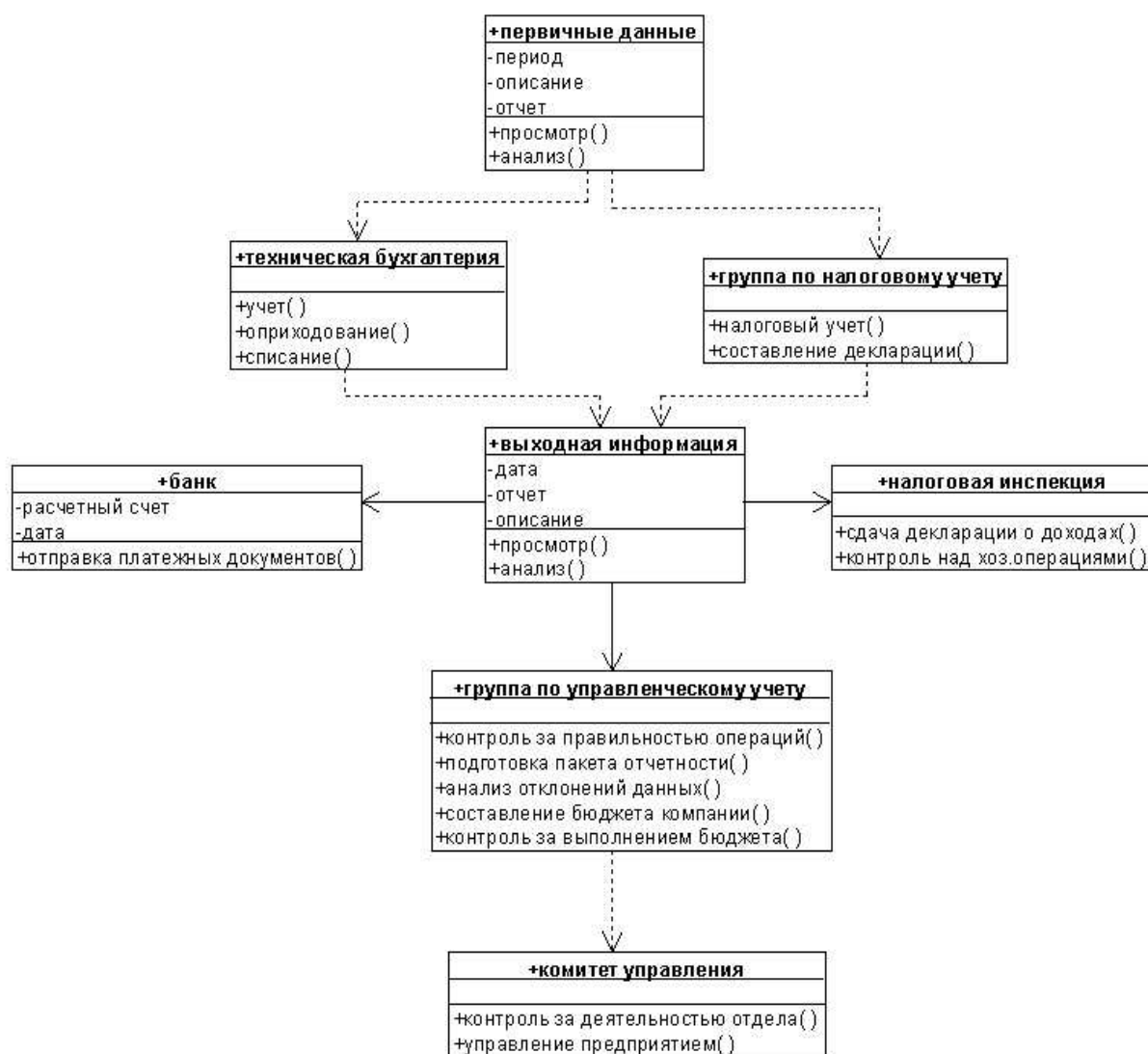


Рисунок 133 – Диаграмма классов

Для рассмотрения взаимодействия объектов в контексте статической структуры модели и для представления структурных особенностей передачи и приема сообщений между объектами используется диаграмма кооперации (рис. 134). Работа в системе осуществляется в следующем порядке: первичные данные, поступая в финансовый отдел, учитываются

технической бухгалтерией, часть из них обрабатывается для отчета, другая поступает в группу по налоговому учету, где формируется отчетность для налоговой службы. Из технической бухгалтерии и группы по налоговому учету данные поступают в группу по управленческому учету, где происходит подготовка пакета отчетности на основе полученных данных и составляется бюджет компании.

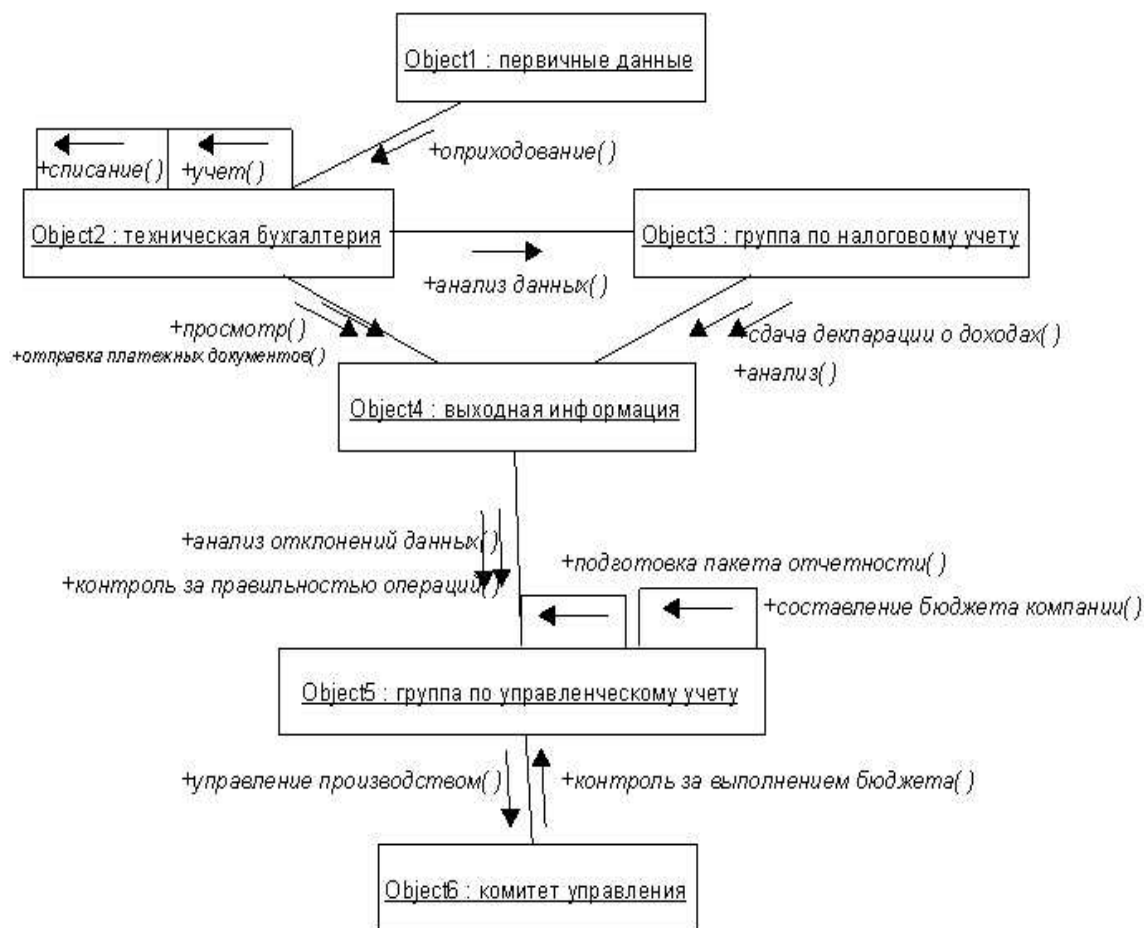


Рисунок 134 – Диаграмма коопераций

Класс «Комитет управления» осуществляет контроль над выполнением бюджета и вносит изменения в управления производством непосредственно через связь с классом «Группа по управленческому учету».

Для моделирования взаимодействия объектов во времени используется диаграмма последовательности (рис. 135). На ней видно, что инициатором взаимодействия является само производство, которое является источником первичных данных и которое после их обработки в финансовом отделе подвергается влиянию комитета управления.

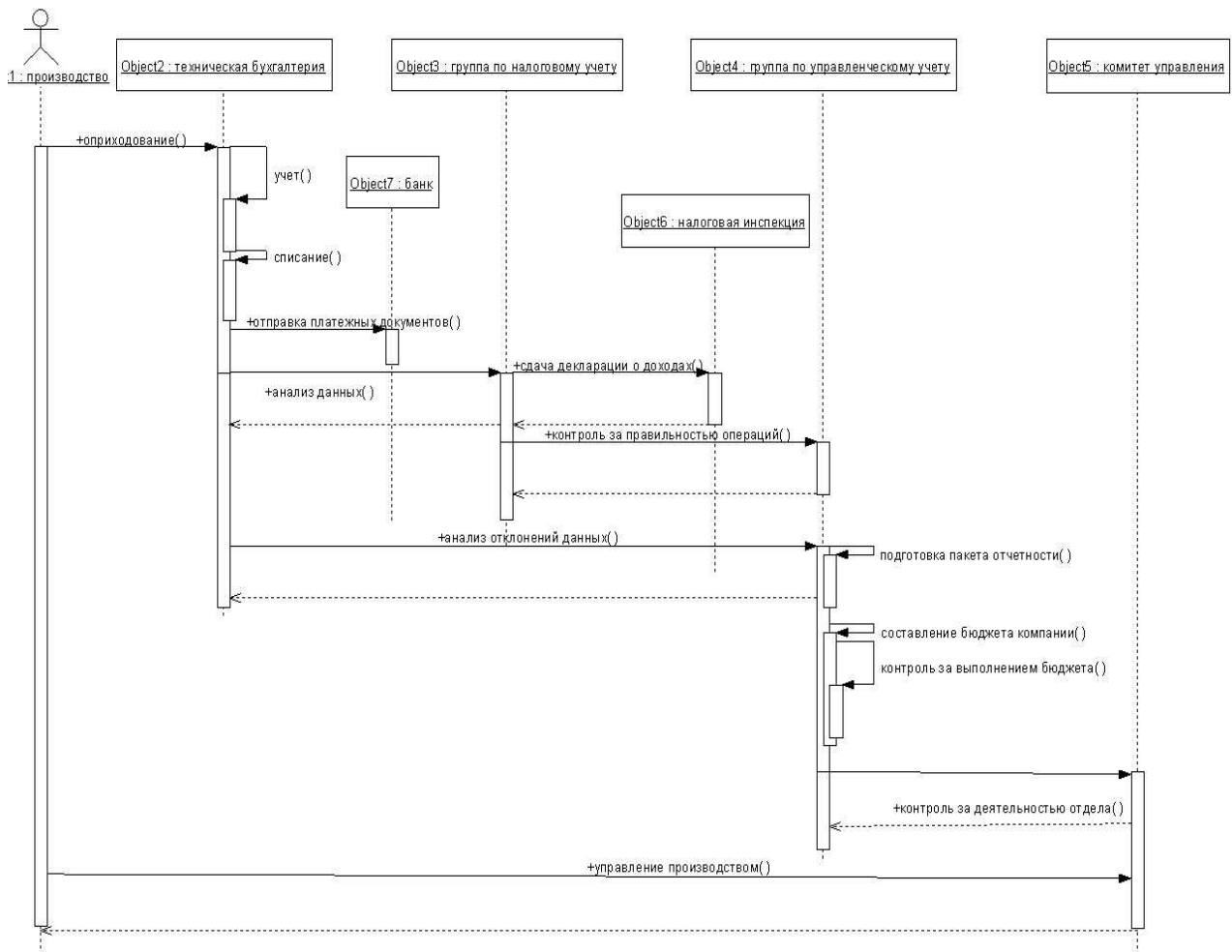


Рисунок 135 – Диаграмма последовательности

Диаграмма состояний описывает все возможные состояния одного экземпляра определенного класса или всей системы и возможные последовательности его (ее) переходов из одного состояния в другое. Диаграмма состояний работы всей системы показана на рис. 136.

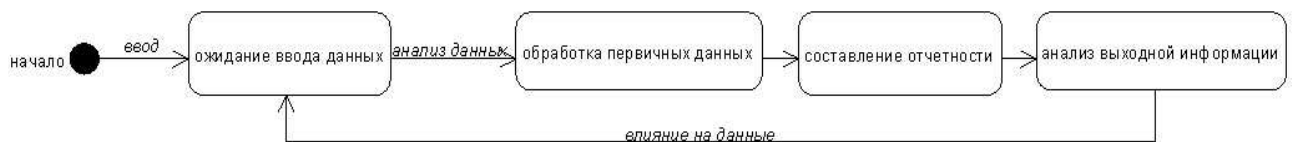


Рисунок 136 – Диаграмма состояний работы системы

Для моделирования процесса выполнения операций используется диаграмма деятельности (рис. 137). На ней показано, что после формирования начальных данных на производстве совершается их детальная обработка во всех трех группах финансового отдела. После чего обрабо-

танные данные поступают комитету управления, где ведется полный контроль над деятельностью не только отдела, но и всего предприятия в целом.

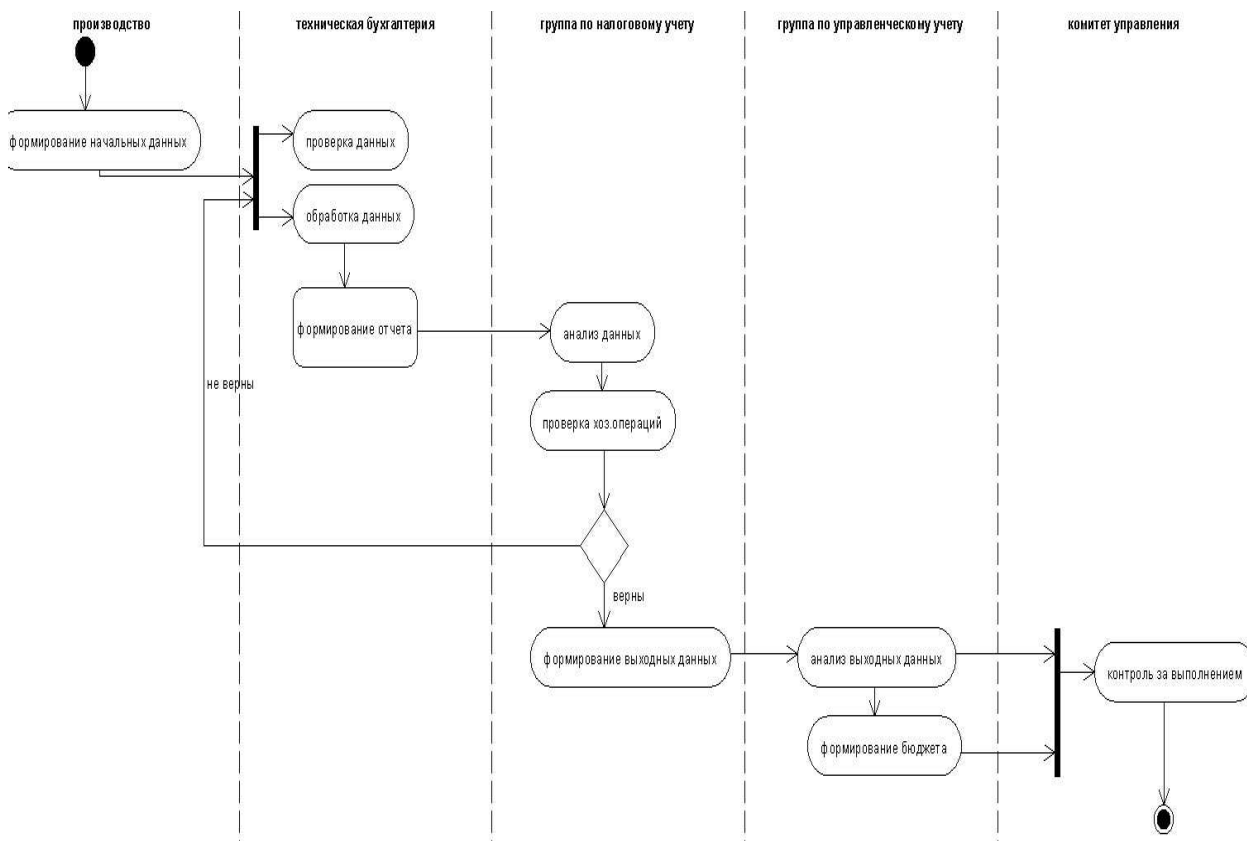


Рисунок 137 – Диаграмма деятельности

Задача разработки программного комплекса не ставилась, поэтому диаграммы реализации не составлялись.

4.6 Информационная система для расчета себестоимости металлопродукции

Себестоимость продукции зависит от условий производства и реализации продукции. Существенное влияние на уровень затрат оказывают технико-экономические факторы производства – изменения в технике, технологии, организации производства, в структуре и качестве продукции и величине затрат на ее производство. На любом производстве важно правильно спланировать ожидаемые доходы, учесть все издержки фирмы и правильно отнести их на себестоимость продукции. Правильно

спроектированная система расчета калькуляции может оказать значительное влияние на процесс формирования себестоимости.

Такая система должна иметь грамотную структуру, позволять оптимизировать информационные потоки, отсеивать ненужную информацию, упрощать поиск и получение необходимой информации. Автоматизация, внедряемая искусственным путем в естественные информационные потоки, будет эффективна только тогда, когда произойдет успешная интеграция автоматизированной информационной системы в структуру предприятия.

На ОАО «СКМЗ» активно поддерживается внедрение автоматизации в производственные, управленческие, вычислительные и информационно-аналитические процессы. Предприятием выделяются средства на исследование областей, которые подлежат автоматизации. В середине 90-х годов была разработана и внедрена в планово-экономический отдел информационная система для составления калькуляции и расчета себестоимости металлопродукции. Данная программа существенно снизила трудоёмкость работ и себестоимость выполнения операций. Однако развитие компьютерных технологий требует от любой информационной системы усовершенствования: переход с MS-DOS на Windows затруднил функционирование программы, так как она не была предназначена для работы в новых операционных системах. Она стала выдавать ошибки, иногда сопровождающиеся потерями данных. Так что появилась необходимость в создании более новой и надежной версии программы, которая смогла бы обеспечить надежность в работе.

Создание системы начинается с проектирования ее модели на языке UML. Сначала необходимо четко определить требования, которым должна соответствовать разрабатываемая система, и какие задачи она должна решать. Разрабатываемая система должна:

- а) реализовывать функции ввода, модификации и просмотра данных;
- б) выполнять основную функцию – расчет себестоимости;
- в) реализовывать функцию составления отчетности полученных данных;
- г) предоставлять простой удобный интерфейс для работы с данными;
- д) иметь возможность сохранения и редактирования данных;
- е) иметь защиту.

Диаграмма вариантов использования представлена на рис. 138.

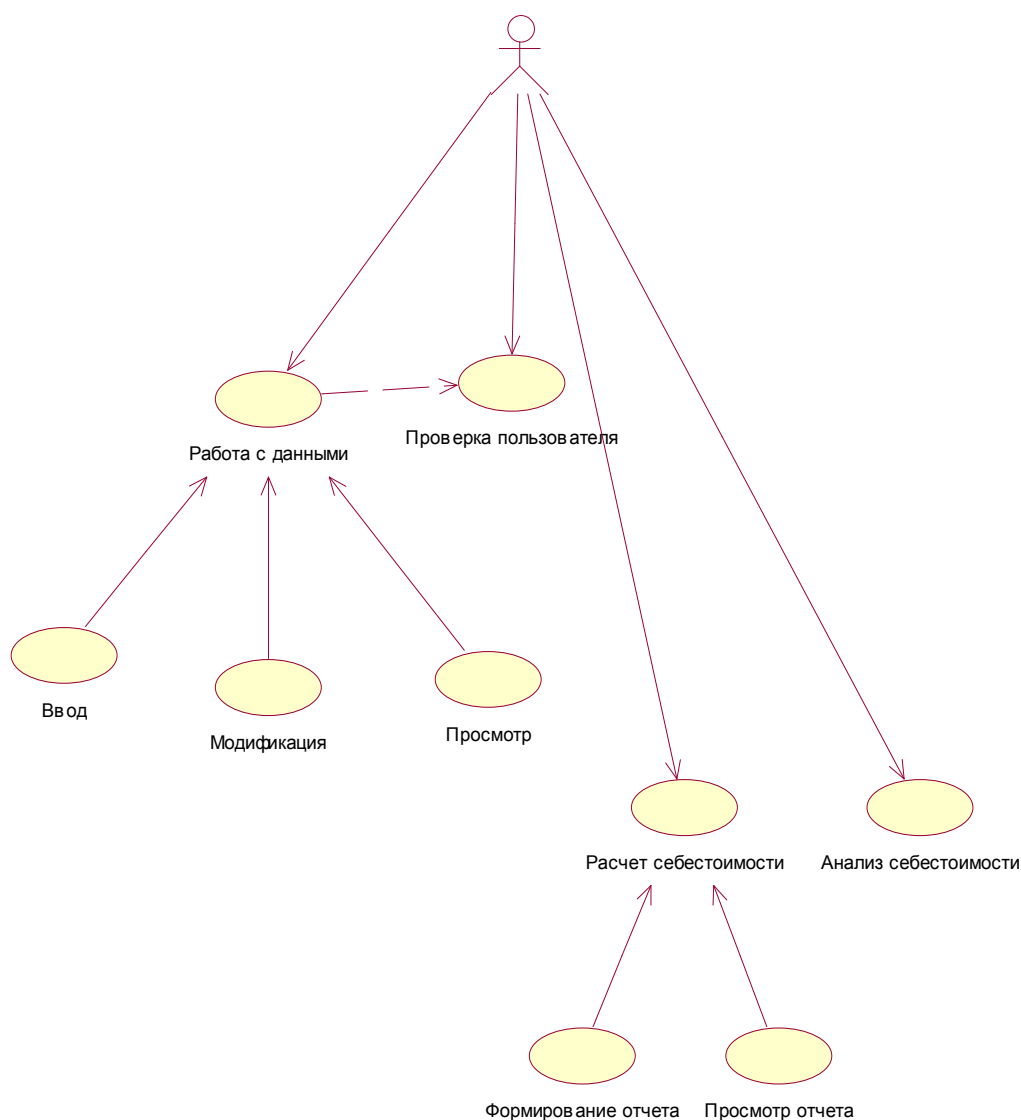


Рисунок 138 – Диаграмма вариантов использования

Непосредственным пользователем программы в данном случае является специалист планово-экономического отдела. На диаграмме видно, что предполагается наличие таких вариантов использования системы: ввод данных, составление калькуляции расчет и анализ себестоимости, экспорт полученных данных. Работа с исходными данными предполагает также их просмотр и модификацию.

Анализ структуры и функционирования системы проводится с помощью диаграмм классов и поведения. Диаграмма классов для расчета себестоимости металлопродукции представлена на рис. 139.

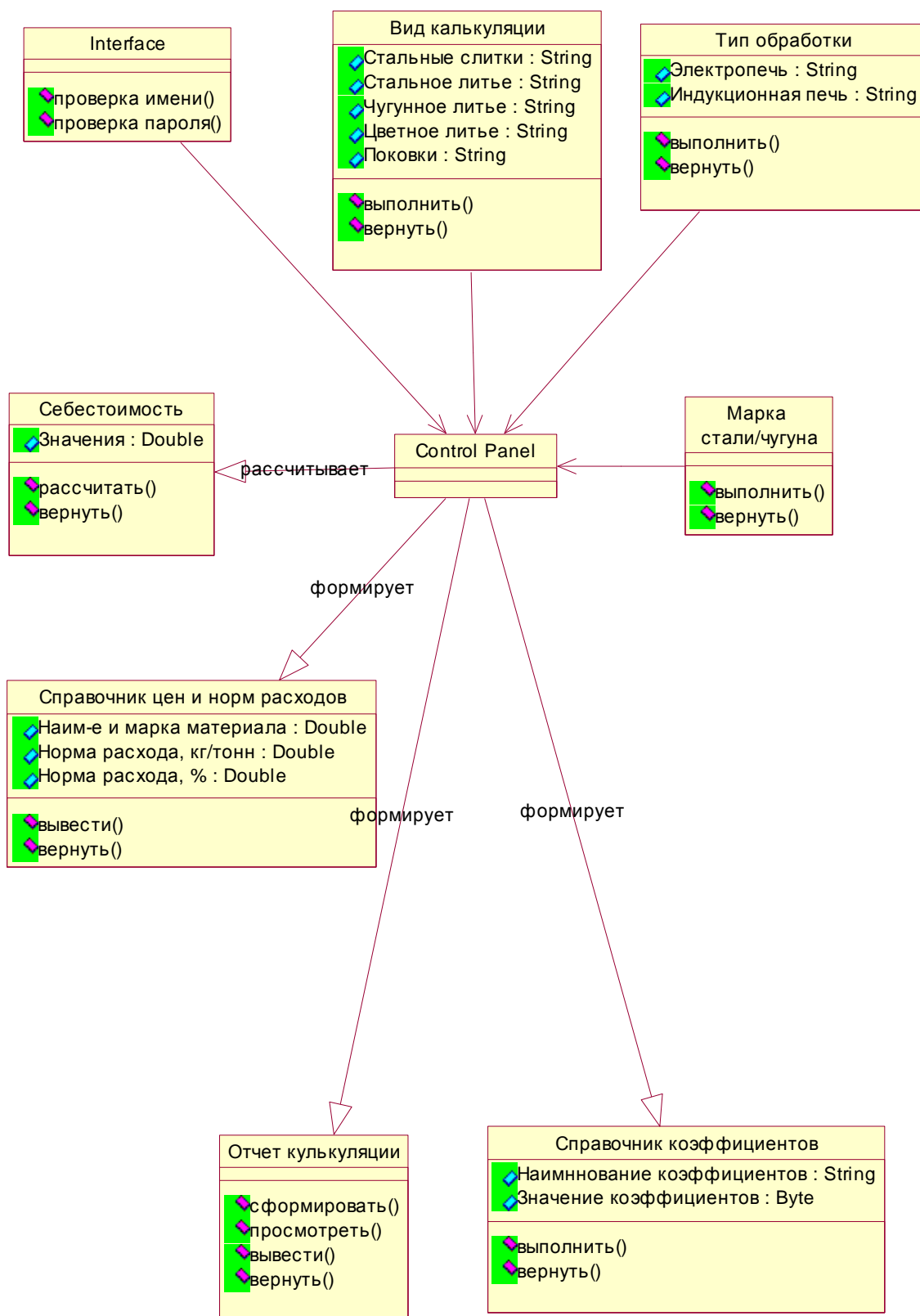


Рисунок 139 – Диаграмма классов

На этой диаграмме изображены следующие классы: interface, вид калькуляции, тип обработки, вид марки стали, справочник коэффициентов, отчет калькуляции, справочник цен и норм расходов, себестоимость, control panel. Все операции классов характеризуются областью видимости типа общедоступный (private), т.е. они видны и доступны из любого другого класса. Control panel является управляющим классом и отвечает за функционирование других классов. Стрелки с треугольниками на схеме обозначают наличие отношений и бинарной связи между классами. Бинарная связь означает, что взаимодействие между классами свободное.

К диаграммам поведения относятся диаграммы коопераций, последовательностей, состояния и деятельности. Диаграмма кооперации представлена на рис. 140. На ней представлены такие классы: interface, вид калькуляции, обработка, марка стали/чугуна, справочник коэффициентов, справочник цен и норм расходов, отчет по калькуляции, себестоимость. Работа системы имеет определенный порядок. Экономист начинает работу окном управления, предварительно введя своё имя и пароль. Затем вводит все необходимые данные для расчета себестоимости путем выбора вида калькуляции, типа обработки, марки стали или чугуна. После, через окно управления, дает команду на формирование справочников коэффициентов, цен и норм расходов, создает отчет по калькуляции. Завершающим этапом является расчет себестоимости металлопродукции.

Для моделирования взаимодействия объектов во времени в языке UML используются диаграммы последовательности (рис. 141). На диаграмме показано, что инициатором взаимодействия является экономист-пользователь, который ведет непосредственное управление и контроль над системой на всем протяжении её работы. Постепенно управление переносится на окно управления, которое собирает необходимую информацию с одних классов (Interface, Вид калькуляции, Обработка, Марка чугуна/стали) и дает команды выполнения другим классам (Справочник коэффициентов, Справочник цен и норм расходов, Отчет калькуляции, Себестоимость). В конце работы создаются необходимые отчеты, и рассчитывается калькуляция, после чего управление передается экономисту.

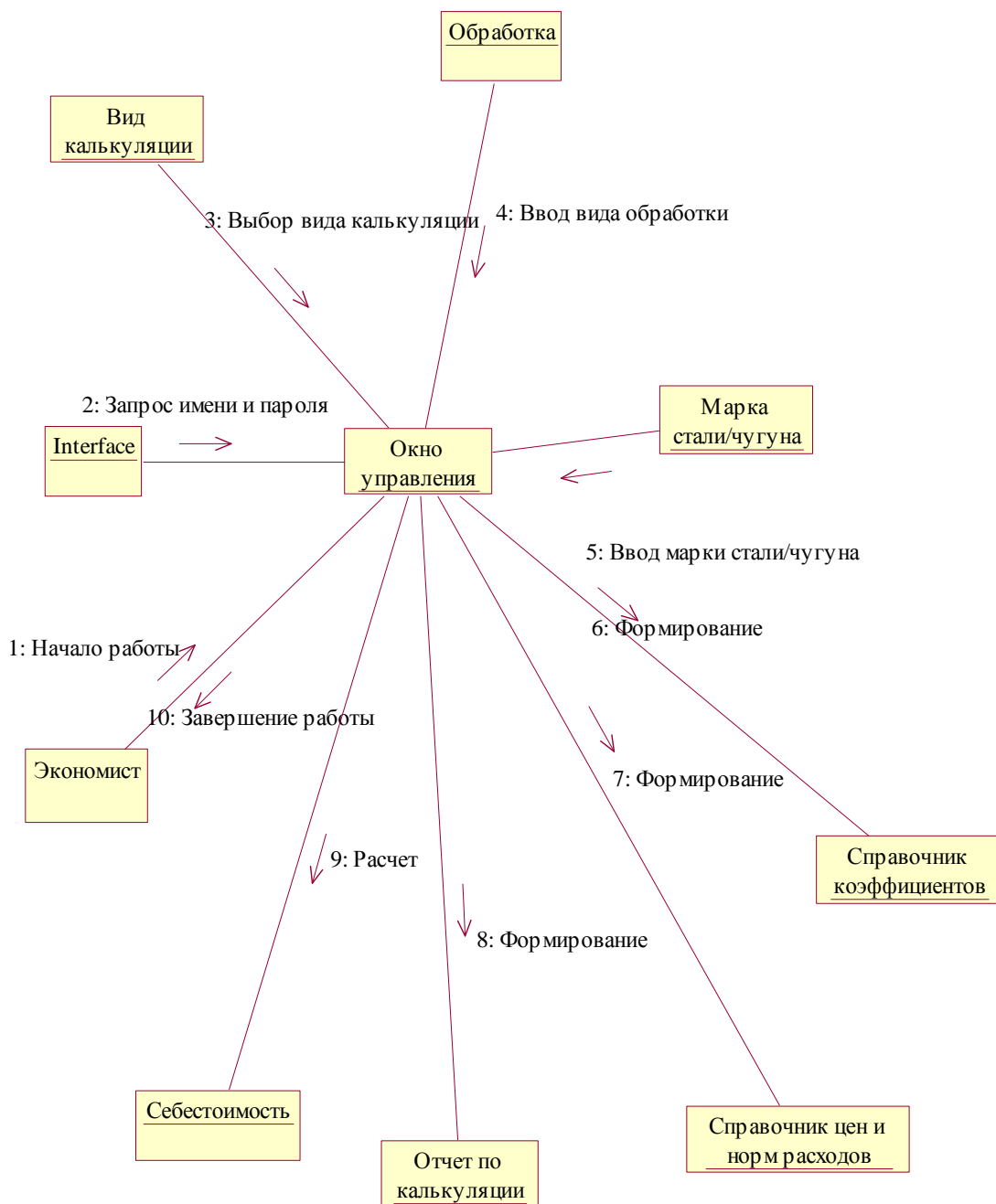


Рисунок 140 – Диаграмма кооперации

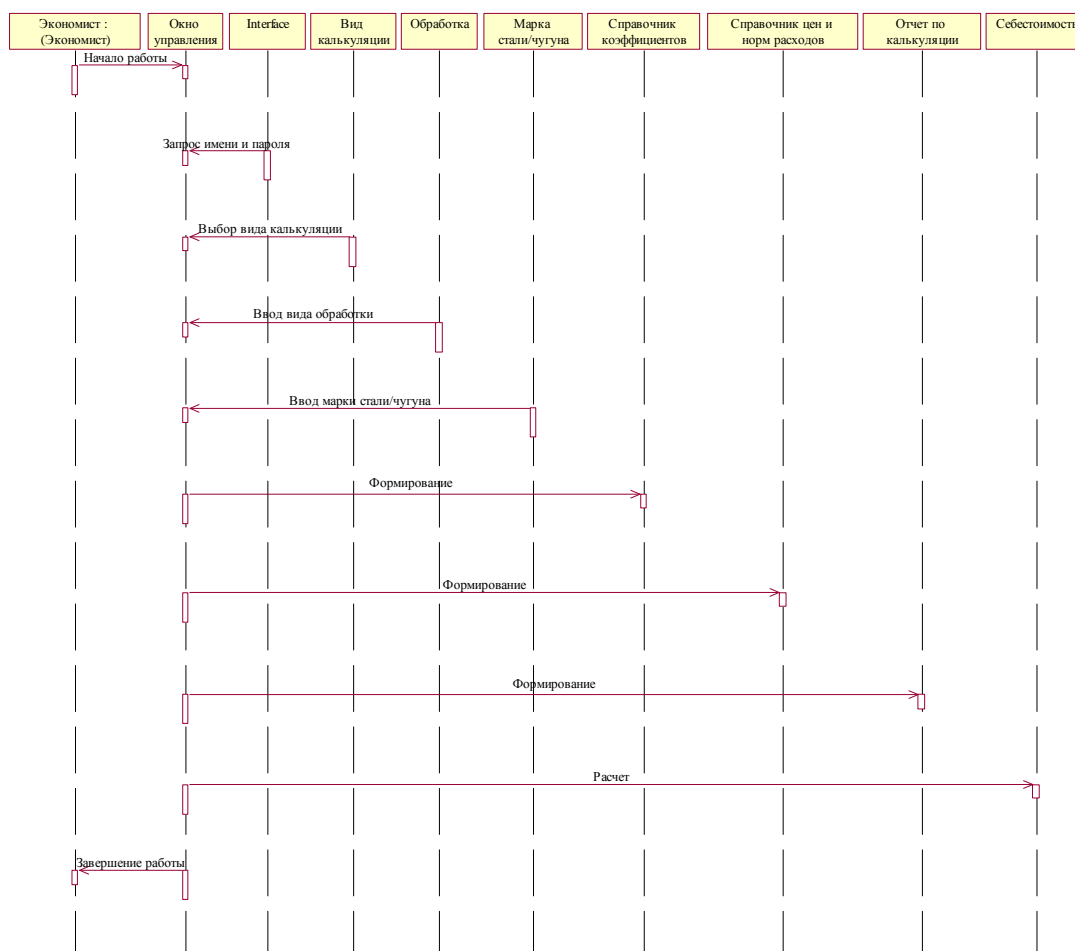


Рисунок 141 – Диаграмма последовательности

Функционирование системы характеризуют диаграммы состояний (рис. 142) и деятельности (рис. 143). Работу системы можно описать так: после обращения экономиста-пользователя система идентифицирует его путем проверки имени и пароля; если проверка не прошла успешно, предполагается выход из системы; в обратном случае экономист получает доступ к главному окну управления. Дальнейшая работа заключается в выборе вида калькуляции (на главном меню), типа обработки, выбора марки стали/чугуна. После этого предполагается сформировать и соотнести на калькуляцию справочник цен и норм расходов, справочник коэффициентов, выполнить расчет себестоимости и затем выполнить сохранение расчетов в файл.

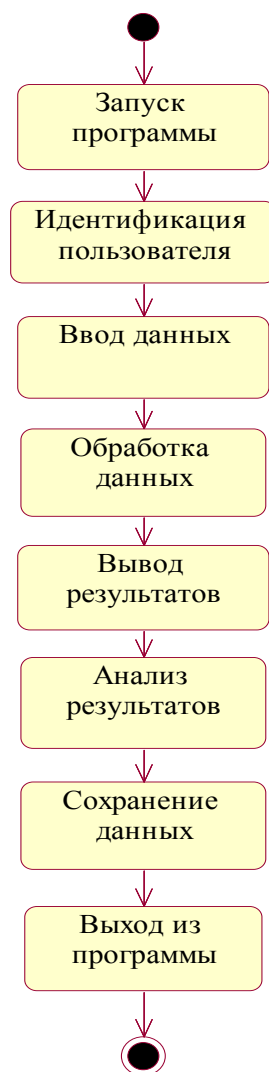


Рисунок 142 – Диаграмма состояний

Для физического представления моделей систем используются диаграммы компонентов и развертывания.

Диаграмма компонентов (рис. 144) показывает, что программная система состоит из таких физических частей: exe, pas, csv, dbf и xls. Компонента проекта Project1.exe выполняет роль основы системы и обеспечивает взаимодействие всех компонентов, файлы pas содержат программный код, файлы dbf – информационные данные.

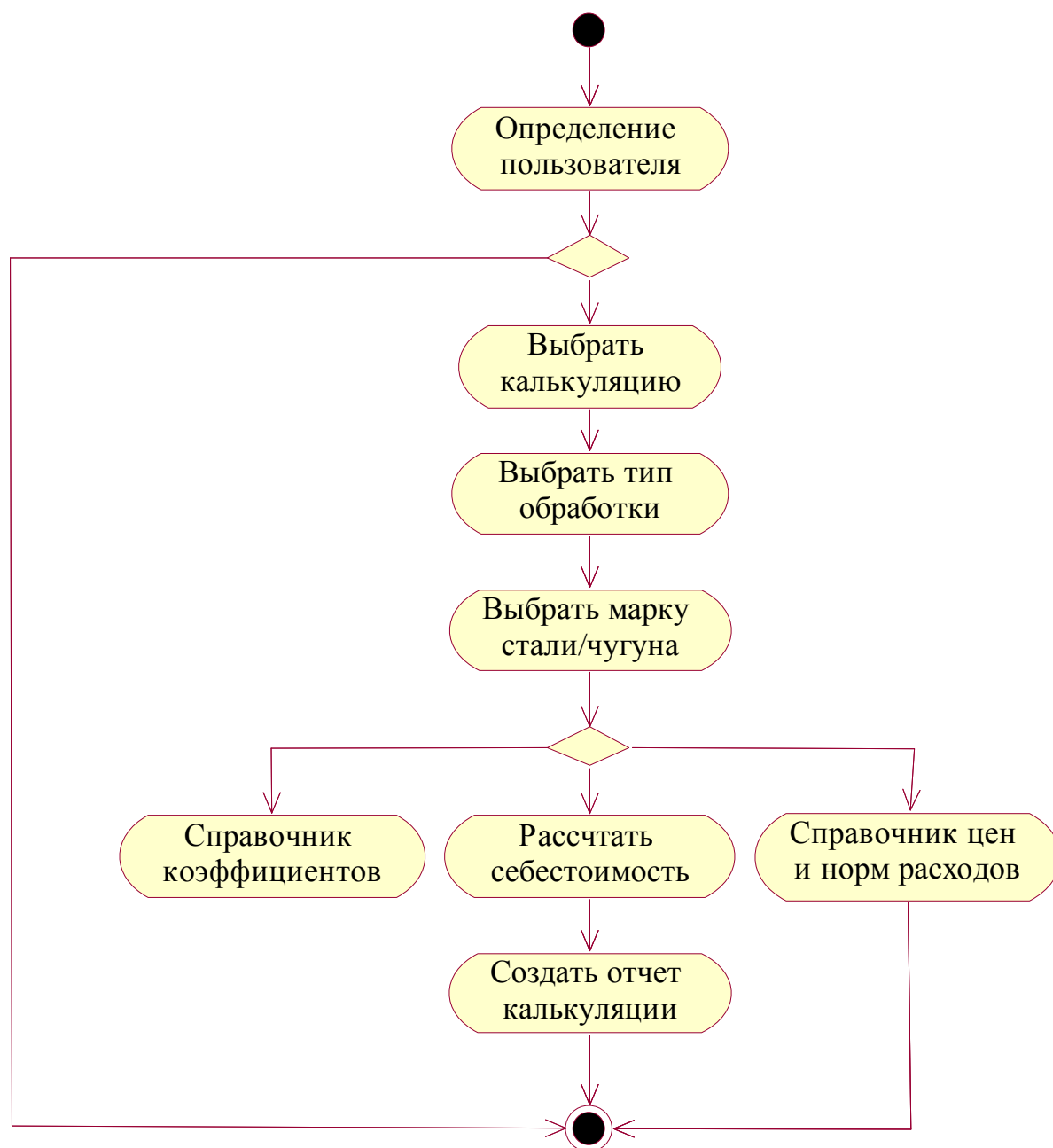


Рисунок 143 – Диаграмма деятельности

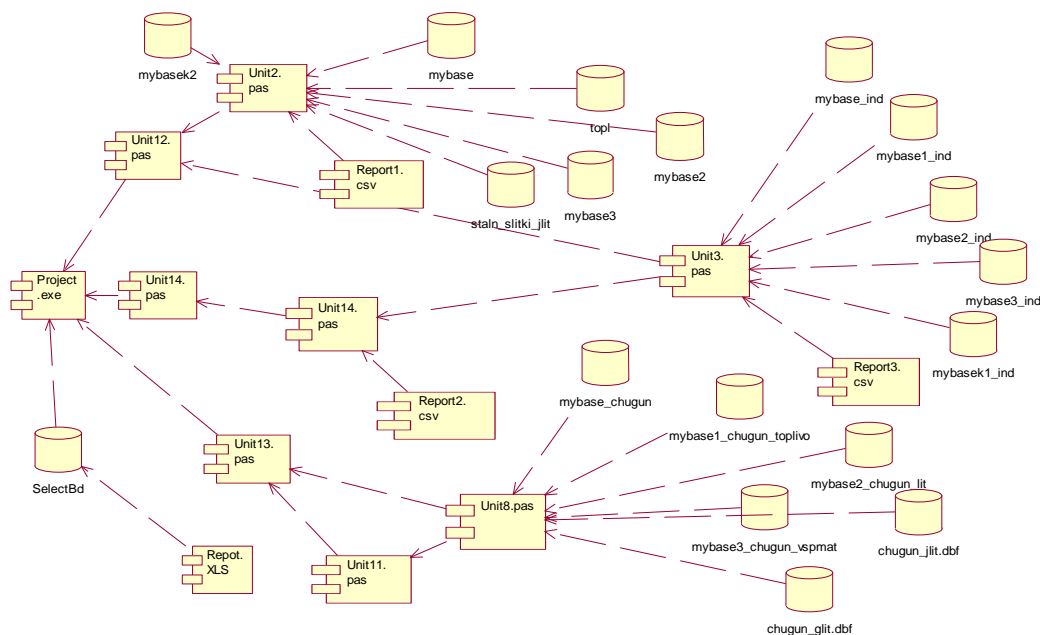


Рисунок 144 – Диаграмма компонентов

Диаграмма развертывания содержит графические изображения процессоров, устройств, процессов и связей между ними. Поскольку в нашем случае разрабатывается программа, которая не будет использовать периферийные устройства и ресурсы (ее функционирование ограничено на компьютере пользователя), поэтому необходимость в разработке этой диаграммы отсутствует.

Созданная модель была реализована в среде Borland-Delphi 6.0 (программа «Калькуляция 1.1») и используется на предприятии для расчета себестоимости металлопродукции, которую оно производит (стальное литье, стальные слитки, чугунное литье, цветное литье, поковки). Программа работает в операционных системах Windows XP/2000/ME, характеризуется хорошей производительностью и потребностью в минимальных системных требованиях.

Был рассчитан годовой экономический эффект от внедрения системы: он составил 7711,8 грн. На одну гривну единовременных капиталовложений величина годового прироста прибыли составляет 0,9 грн. Срок окупаемости составил 1 год и 1 месяц [26-27].

4.7 Информационная система для учета и контроля готовой продукции

Основными задачами учета готовой продукции на предприятии являются: своевременное оформление соответствующими документами готовой продукции, выпущенной на производстве; обеспечение контроля за ее сохранностью на складах предприятия; своевременное отражение операций по отгрузке и реализации готовой продукции и расчетов с покупателями; обеспечение контроля за выполнением плана выпуска и реализации продукции. Для повышения точности расчетов и снижения трудоемкости такой деятельности ее необходимо автоматизировать.

Проблема автоматизации учета и контроля готовой продукции остро стоит на многих предприятиях Украины, в том числе на КиАЗ «Авиант». Существует несколько путей решения проблемы: а) проводить расчеты и составлять документацию вручную с использованием средств Microsoft Office; б) использовать разработанные ранее программные продукты; в) использовать приобретенные программные продукты; г) разрабатывать собственные программные продукты для решения поставленных задач.

Проведение расчетов и составление документации вручную приведут к большим затратам труда и времени работников, выполняющих учет готовой продукции и полуфабрикатов. Использование корпоративной информационной системы (например, «1С-предприятие») не может решить проблему: даже многопользовательской версии «1С» недостаточно для такого предприятия, как КиАЗ «Авиант»; для решения проблемы масштабируемости придется покупать несколько лицензий, а это приведет к большим материальным затратам.

В начале 90-х годов на Киевском авиационном заводе «Авиант» сотрудниками отдела автоматизированных систем управления производством для решения задачи учета и контроля готовой продукции был разработан программный продукт, который называется «ГИЗы». В настоящее время эта программа требует значительной доработки – перевода на новую вычислительную платформу, обеспечение многопользовательского сетевого режима, приведение интерфейса к современным стандартам

и многое другое. Очевидно, что целесообразно разработать новую программную систему.

Проектирование системы начинается с построения концептуальной модели – диаграммы вариантов использования («прецедентов»). Такая диаграмма для системы учета и контроля приведена на рис. 145.

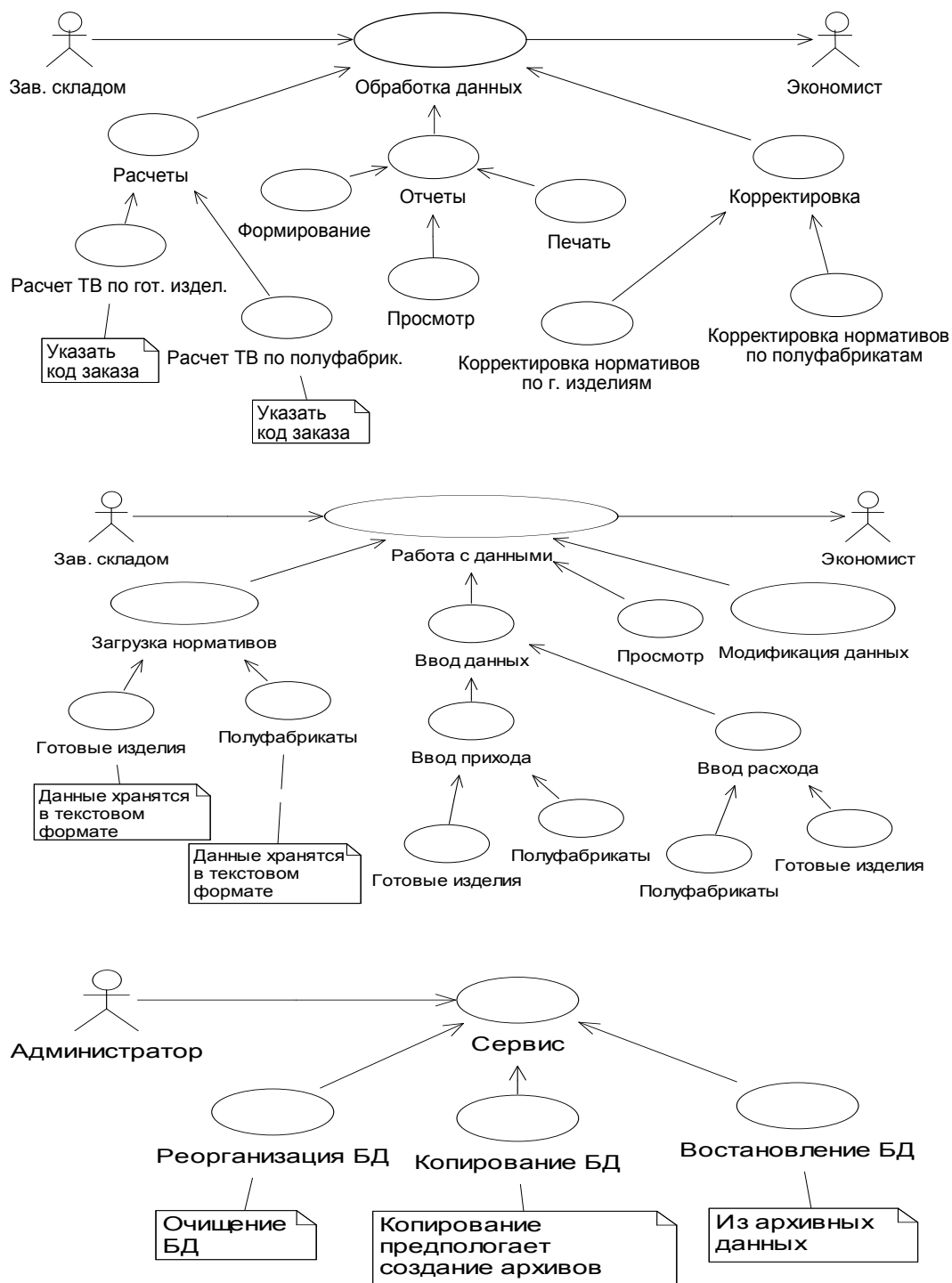


Рисунок 145 – Диаграмма вариантов использования

Центральную диаграмму логической модели системы – диаграмму классов – для удобства отображения разделим на две: диаграмму ассоциаций классов (рис. 146) и диаграмму детализаций классов (рис. 147).

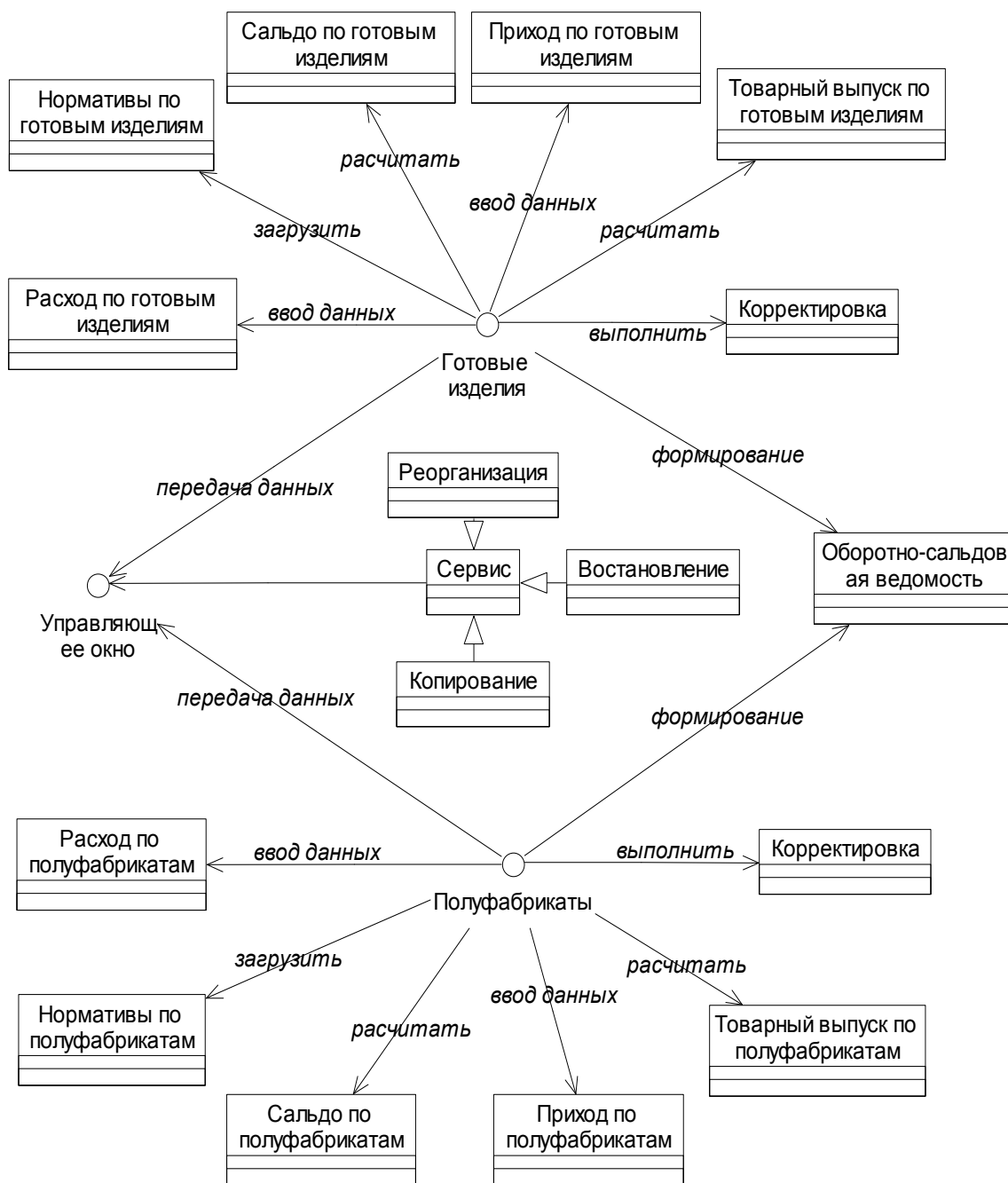


Рисунок 146 – Диаграмма ассоциаций классов

Нормативы по готовым изделиям Год отчетного периода Месяц отчетного периода Балансовый счет Шифр аналитического учета Номер цеха по заказу Количество Цена +загрузка() +корректировка()	Расход по готовым изделиям Год отчетного периода Месяц отчетного периода Номер документа Цех Балансовый счет Шифр аналитического учета Вид операции Количество Цена Сумма +ввод данных() +просмотр данных()	Товарный выпуск по полуфабрикатам Год отчетного периода Месяц отчетного периода Балансовый счет Шифр аналитического учета Балансовый счет заказа Цех Шифр материала Количество Цена +расчет() +просмотр() +печать()
Приход по готовым изделиям Год отчетного периода Месяц отчетного периода Номер документа Цех Балансовый счет Шифр аналитического учета Количество Цена +ввод данных() +просмотр данных()	Сальдо по полуфабрикатам Год отчетного периода Месяц отчетного периода Балансовый счет Шифр аналитического учета Балансовый счет заказа Цех Шифр материала Количество Цена Сумма +расчет() +просмотр() +печать()	Сальдо по готовым изделиям Год отчетного периода Месяц отчетного периода Балансовый счет Шифр аналитического учета Балансовый счет заказа Цех Шифр материала Количество Цена Сумма +расчет() +просмотр() +печать()
Приход по готовым изделиям Год отчетного периода Месяц отчетного периода Номер документа Цех Балансовый счет Шифр аналитического учета Количество Цена +ввод данных() +просмотр данных()	Расход по полуфабрикатам Год отчетного периода Месяц отчетного периода Номер документа Цех Балансовый счет Шифр аналитического учета Вид операции Количество Цена Сумма +ввод данных() +просмотр данных()	Сальдо-оборотная ведомость Год отчетного периода Месяц отчетного периода Номер заказа Шифр материала Входящее сальдо Приход Товарный выпуск Исходящее сальдо Количество Цена Сумма +расчет() +просмотр() +печать()
Приход по полуфабрикатам Год отчетного периода Месяц отчетного периода Номер документа Цех Балансовый счет Шифр аналитического учета Количество Цена +ввод данных() +просмотр данных()	Товарный выпуск по готовым изделиям Год отчетного периода Месяц отчетного периода Балансовый счет Шифр аналитического учета Балансовый счет заказа Цех Шифр материала Количество Цена +расчет() +просмотр() +печать()	

Рисунок 147 – Диаграмма детализаций классов

Поскольку взаимодействие объектов в случаях работы с готовыми изделиями и полуфабрикатами несколько различается, диаграмму кооперации представим в двух вариантах – для работы с интерфейсами «Готовые изделия» (рис. 148) и «Полуфабрикаты» (рис. 149).

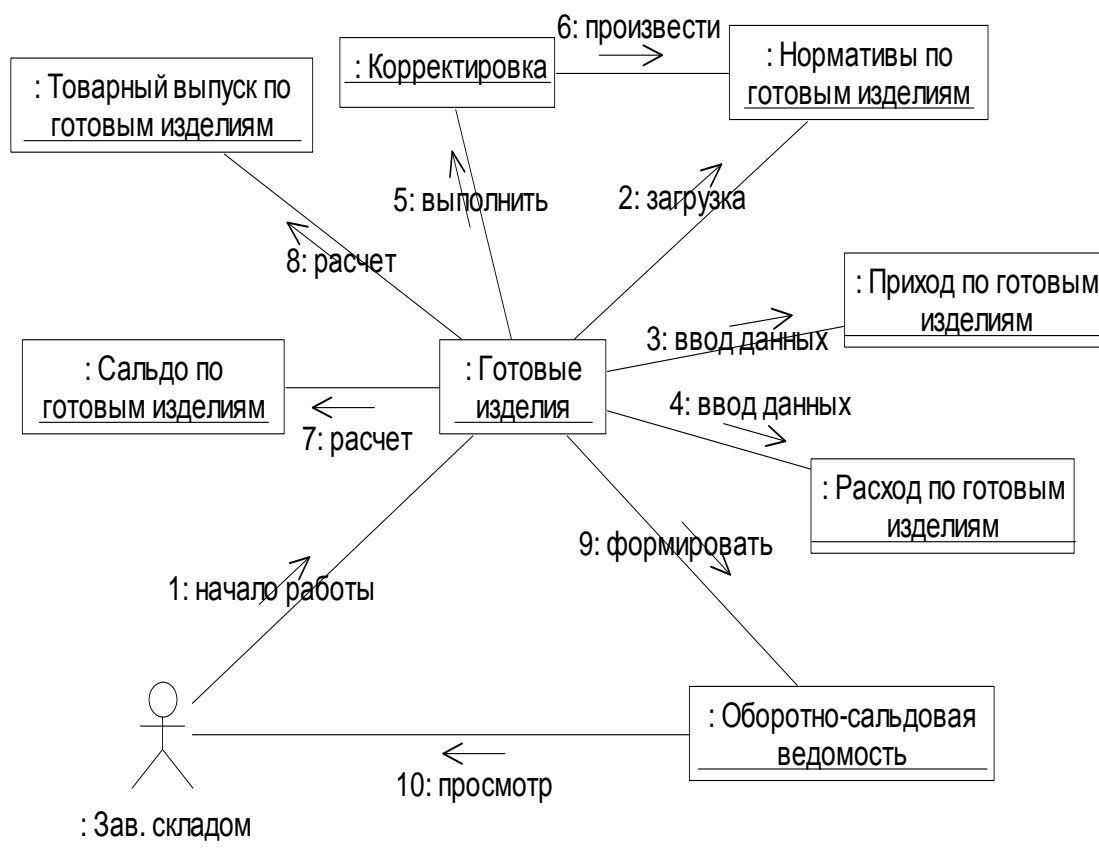


Рисунок 148 – Диаграмма кооперации при использовании интерфейса «Готовые изделия»

Работа нашей системы ничем не отличается от функционирования типовой системы, поэтому диаграмма состояний не строилась.

Для моделирования процесса выполнения операций в языке UML используются диаграммы деятельности; на рис. 150 показаны действия, которые происходят во время функционирования объекта.

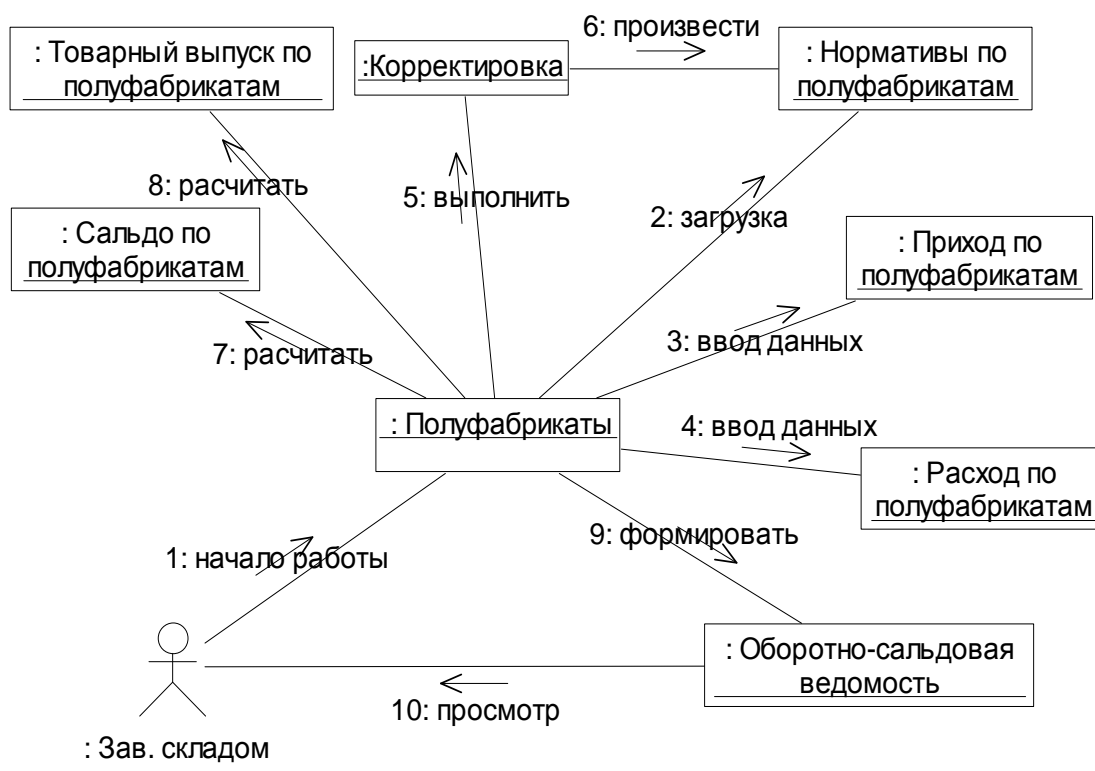


Рисунок 149 – Диаграмма кооперации при использовании интерфейса «Полуфабрикаты»

Перед построением диаграммы компонентов выделим существующие в системе интерфейсы (интерфейс является «лицом» объекта для других объектов системы).

Для работы с данными необходим компонент «Таблица», а для представления данных таблицы – интерфейс «Работа с данными». В свою очередь, работа с данными может включать просмотр и редактирование данных, применение запросов на выборку по заданному условию и формирование отчета по заданному условию. Следовательно, выделим следующие интерфейсы: «Ввод», «Загрузка», «Просмотр», «Редактирование», «Запрос на выборку», «Отчет». Полученная диаграмма компонентов представлена на рис. 151. Компоненты, имеющие тип «::Таблица», на физическом уровне представлены соответствующими таблицами базы данных.

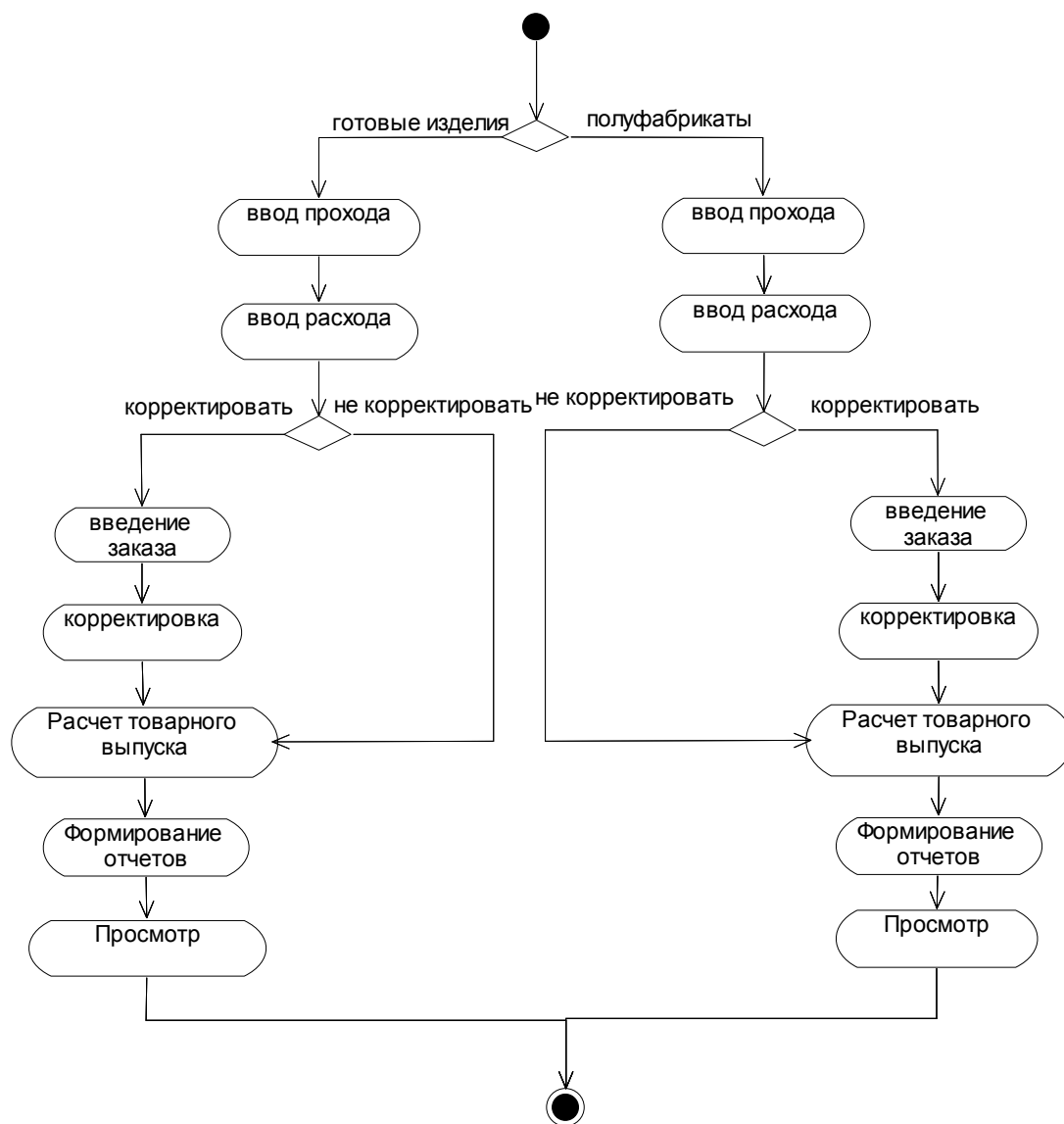


Рисунок 150 – Диаграмма деятельности

Для представления общей конфигурации и топологии распределенной программной системы существует диаграмма развертывания (размещения). Проанализируем узлы, задействованные в нашей системе.

Проектируемая программа должна работать с базой данных. База данных является удаленной, т.е. размещается на сервере сети (архитектура проектируемой «Клиент-сервер»). Для получения данных «клиент-приложение» формирует и отправляет запрос удаленному серверу. Сервер применяет условие запроса к данным и по сети отправляет «клиенту-приложению» соответствующие данные (рис. 152).

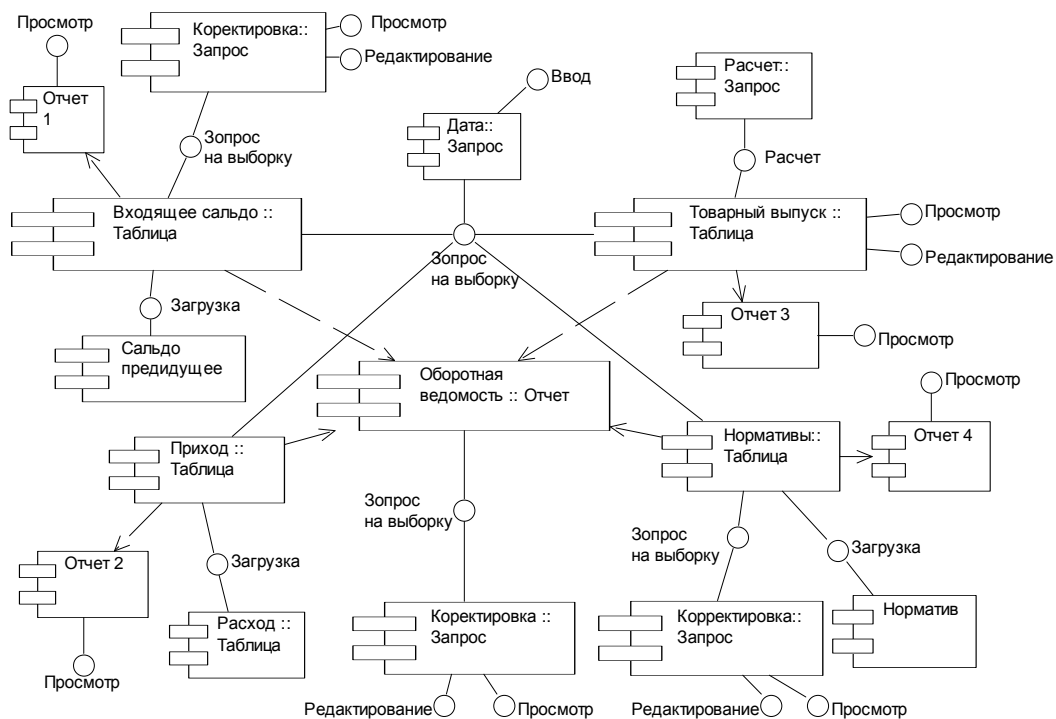


Рисунок 151 – Диаграмма компонентов

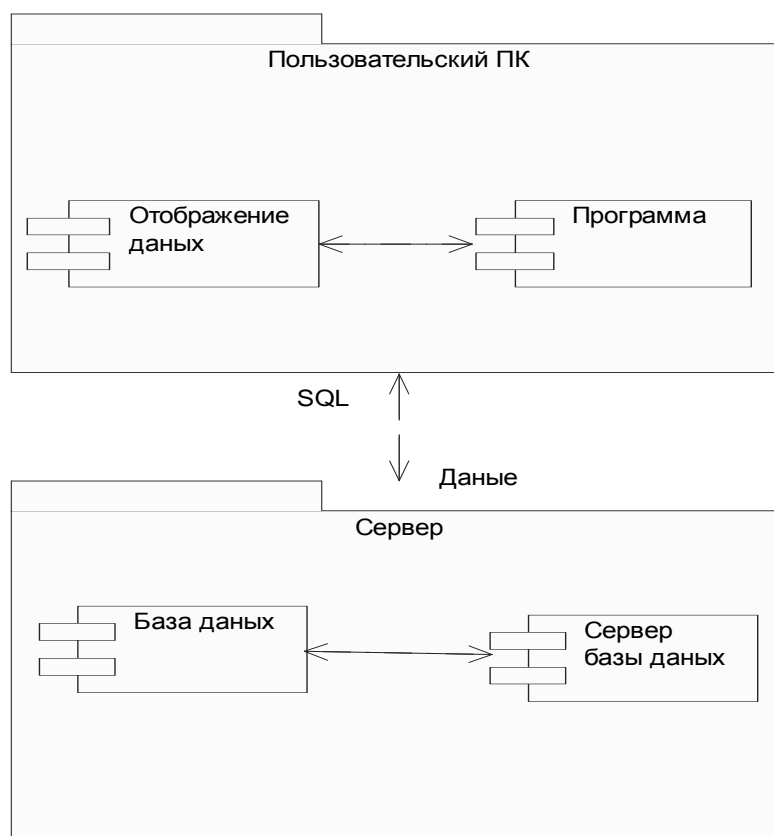


Рисунок 152 – Диаграмма развертывания

Спроектированная информационная система была реализована в среде Borland Delphi 7 и как программный продукт «Готовые изделия» внедрена на Киевском авиационном заводе «Авиант» [28].

Был проведен расчет экономической эффективности, согласно которому прирост прибыли на одну гривну единовременных затрат (капиталовложений) должен составить 5,04 гривны. Срок окупаемости капиталовложений – 2,5 месяца.

4.8 Информационная система для маркетинговых исследований и анализа надежности

Одним из основных этапов создания любого изделия является определение его области функционирования – ниши данного изделия среди множества ему подобных. После создания опытных образцов обычно выполняются лабораторные, а затем и производственные испытания. По результатам испытаний, а также анализа банка данных эксплуатации подобных изделий определяются статистические характеристики надежности объектов. На основании их анализа делается вывод о том, насколько изделие отвечает требованиям надежности, то есть выполняется статистическая оценка одной из важнейших составляющих качества изделий.

Для позиционирования технических изделий и для анализа ниш различных товаров может быть использована информационная система. К примеру, необходимо определить, какая расфасовка стирального порошка будет пользоваться наибольшим спросом. С помощью социального опроса или анкетирования определяется набор данных как вариационный ряд значений и с помощью информационной системы маркетинговых исследований определяется наиболее востребованный объем упаковки.

Таким образом, имеется необходимость удовлетворения потребностей различных субъектов в статистическом анализе применительно к маркетинговым исследованиям для позиционирования изделий и оценки надежности технических объектов. Объект исследования – вариационные ряды числовых значений характеристик использования изделий, а также вариационные ряды времен безотказной работы, времен восста-

новления или наладки. На основе первых проводятся маркетинговые исследования ниш, вторые используются для статистического анализа надежности и контроля качества изделий.

Для проектирования информационной системы был применен унифицированный язык создания моделей – UML (Unified Modeling Language). Он помогает отобразить видение системы и дает возможность обсуждать его со всеми заинтересованными лицами. Это делается с помощью набора обозначений и диаграмм, причем каждая диаграмма играет свою роль в процессе разработки. Построение UML-диаграмм выполнялось в программе Rational Rose Enterprise Edition, поэтому созданные диаграммы отражают особенности этого инструмента.

Сначала сформулируем основные требования к системе, и какие задачи она должна решать:

1. Ввод различного типа данных, их редактирование, конвертирование в соответствующий (при необходимости) и просмотр. Для характеристик объекта могут применяться как вариационные ряды, так и последовательности частот и частотей.
2. Соответствующая обработка введенных данных, сортировка и т.д.
3. Запрос у пользователя необходимых для расчета параметров.
4. Расчет характеристик надежности по введенным данным.
5. Вывод соответствующих графиков и отчетов.
6. Поскольку интерфейс программы должен быть максимально дружелюбен пользователю, информационная система должна иметь оболочку, настраиваемую пользователем.

Визуальное моделирование в UML представляет собой процесс поэтапного спуска от наиболее общей и абстрактной концептуальной модели исходной системы к логической, а затем и к физической модели соответствующей программной системы. Для достижения этих целей вначале строится модель в форме так называемой диаграммы вариантов использования (use case diagram), описывающей функциональное назначение системы.

Непосредственным пользователем системы (актером) является «Пользователь ИС». Ему доступны следующие варианты работы с информационной системой: работа с данными (собственно сами расчеты), работа с формой (просмотр отчетов и информации о системе, настройка интерфейса программы, вычисления на встроенном калькуляторе и др.) и работа с дополнительными модулями – Editor и Converter. Это вполне обособленные программные модули, предназначенные для вспомогательной работы с данными (просмотр данных, их редактирование и конвертирование исходных данных в формат, приемлемый информационной системой). Для упрощения проектирования модель была разбита на 2 модуля (один включает в себя другой). На рис. 153 и 154 приводятся диаграммы этих модулей.

Рисунок 153 – Диаграмма вариантов использования (главная программа)

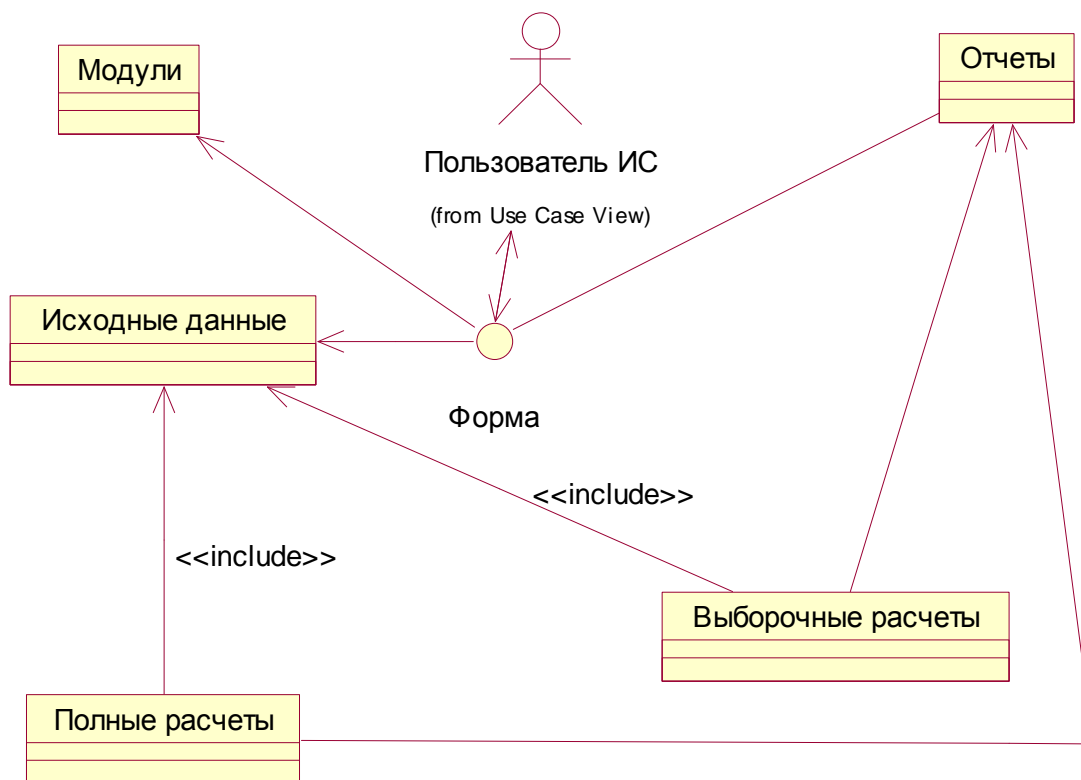


Рисунок 155 – Диаграмма классов

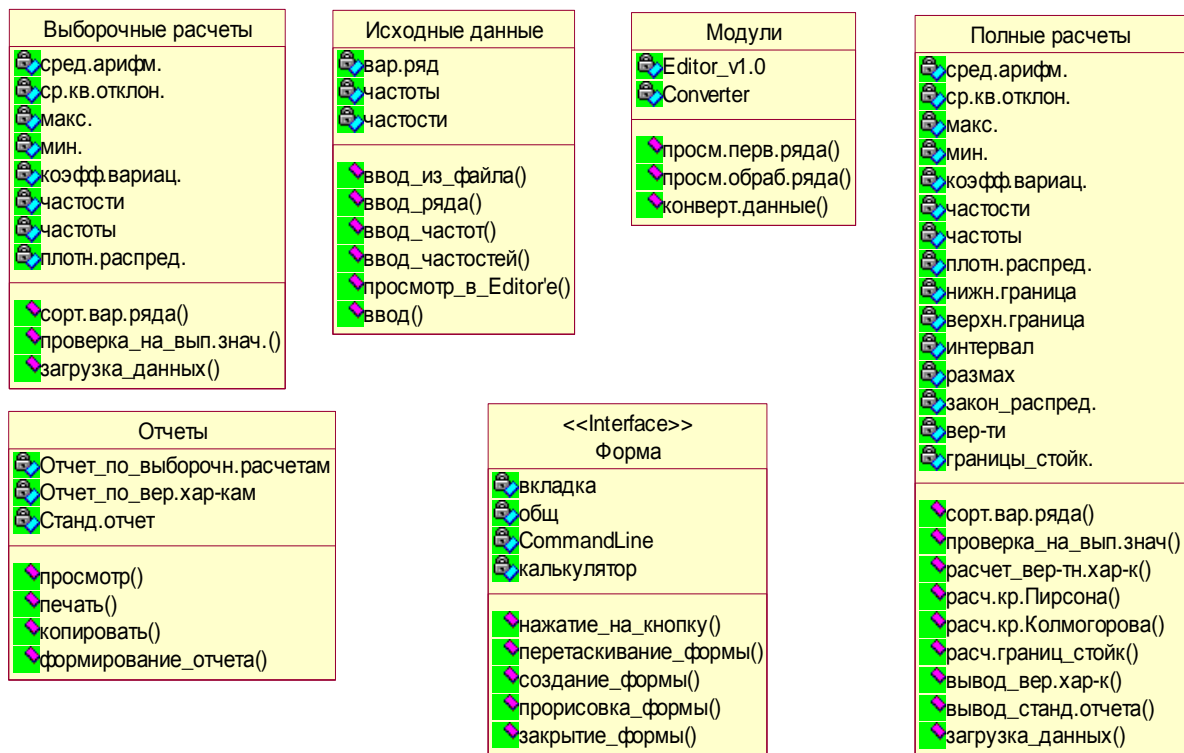


Рисунок 156 – Конкретизация классов

Для отражения динамических особенностей (аспектов) проектируемой системы применяются диаграммы поведения, которые представлены различными видами диаграмм (состояний, деятельности и взаимодействия). В интерпретации Rational Rose диаграммы состояний и деятельности объединены единым названием – State Machine Diagram. Однако построение их взаимно независимо.

Диаграмма состояний (Statechart) предназначена для отображения состояний объектов системы, имеющих сложную модель поведения. Состояния соединены между собой отношениями ассоциации, над которыми, как правило, надписаны условия перехода из состояния в состояние. В проектируемой системе диаграмма состоит из 3 основных состояний: ввод данных, полный и выборочный расчет. Причем, у ввода данных есть подсостояния, а у расчетов есть списки действий (рис. 157).

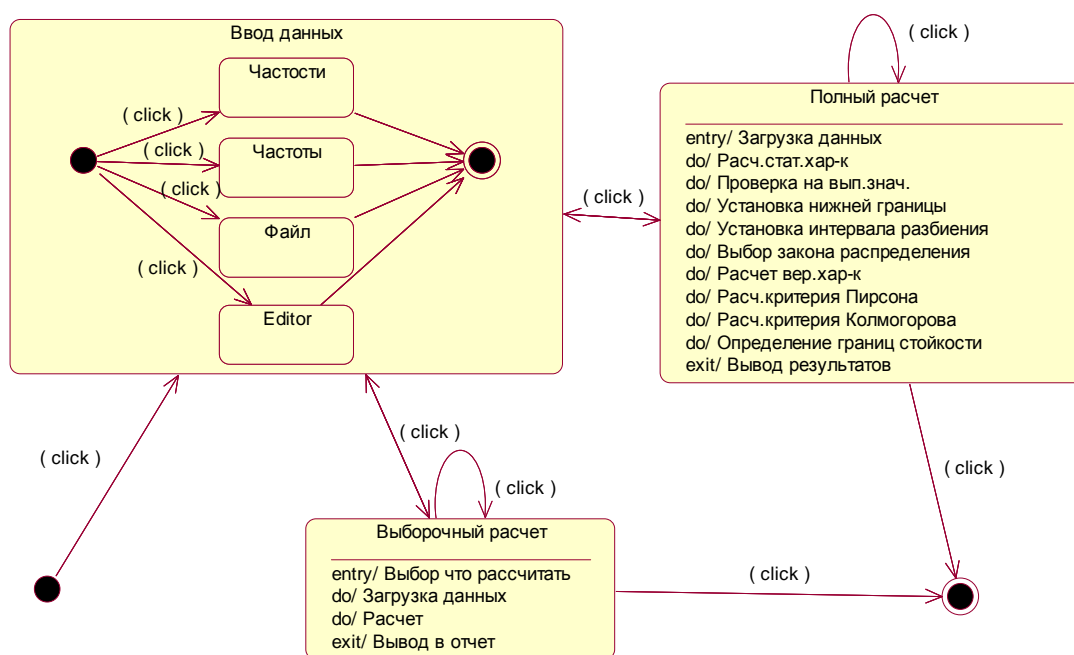


Рисунок 157 – Диаграмма состояний

Диаграмма деятельности – это дальнейшее развитие диаграммы состояний. Она позволяет показать не только последовательность процессов, но и их ветвление и даже синхронизацию. Для построения диаграммы деятельности проектируемой информационной системы были выбраны 4 класса: «Пользователь ИС», «Выборочные расчеты», «Пол-

ные расчеты», «Отчеты» (остальные опущены, как имеющие вспомогательное либо сопутствующее расчетам значение) – рис. 158.

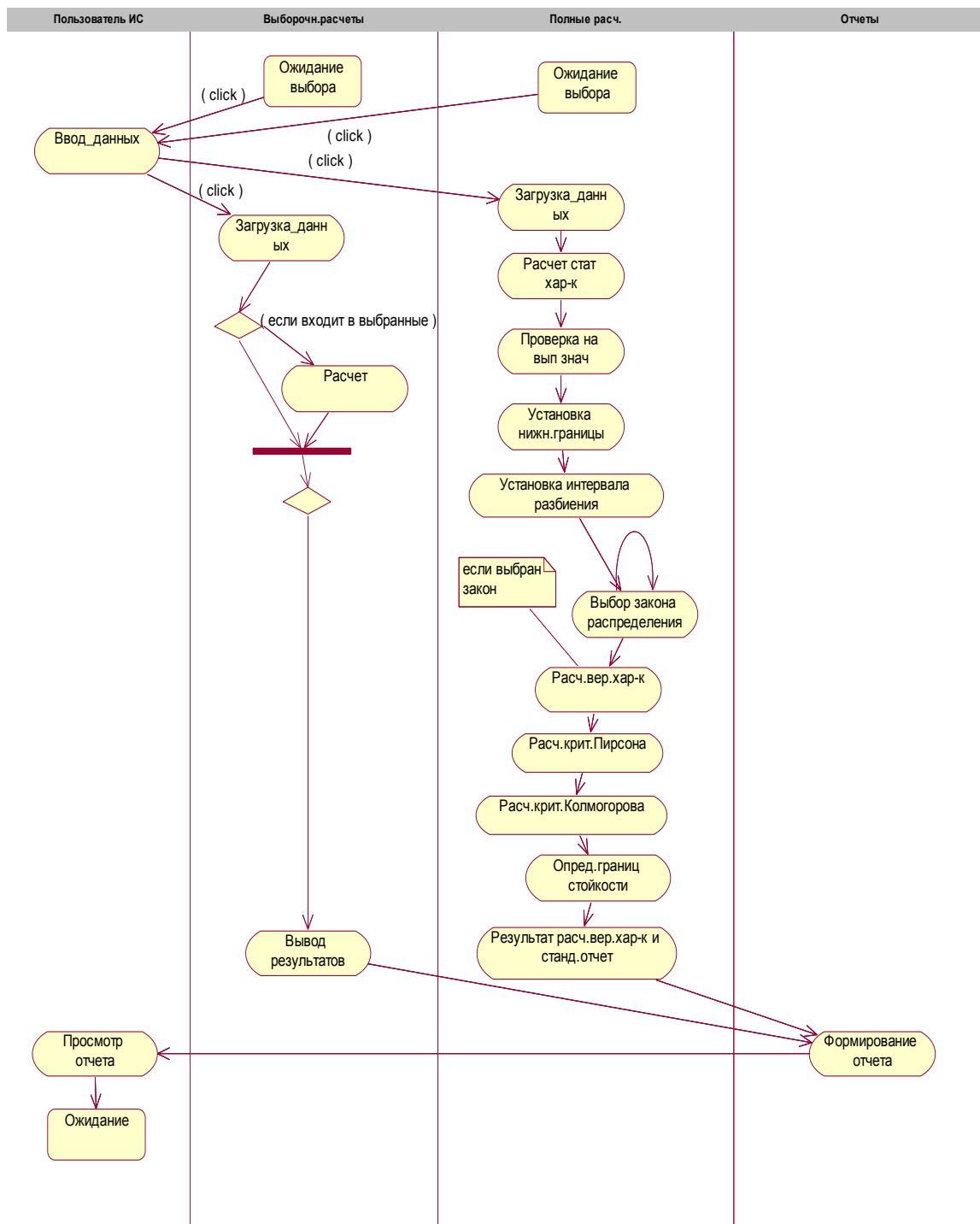


Рисунок 158 – Диаграмма деятельности

Для проектируемой информационной системы можно составить диаграмму кооперации, представленной в 3 аспектах: интерактивная часть (рис. 159) – взаимодействие пользователя системы с различного

вида и назначения формами и отчетами, выборочный расчет (рис. 160) и полный расчет (рис. 161) – взаимодействие пользователя с системой в процессе вышеуказанных расчетов.

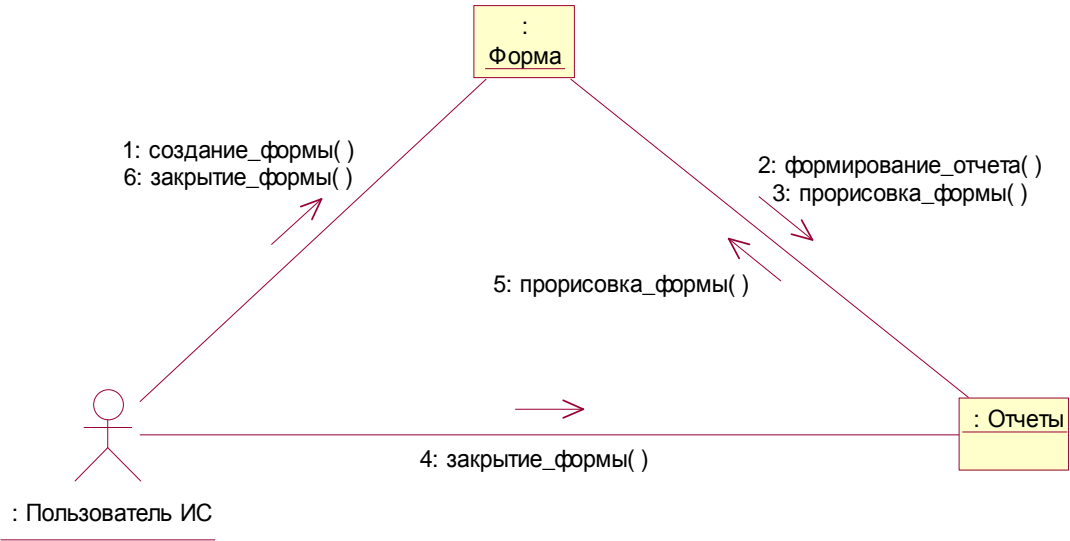


Рисунок 159 – Диаграмма кооперации (интерактивная часть)

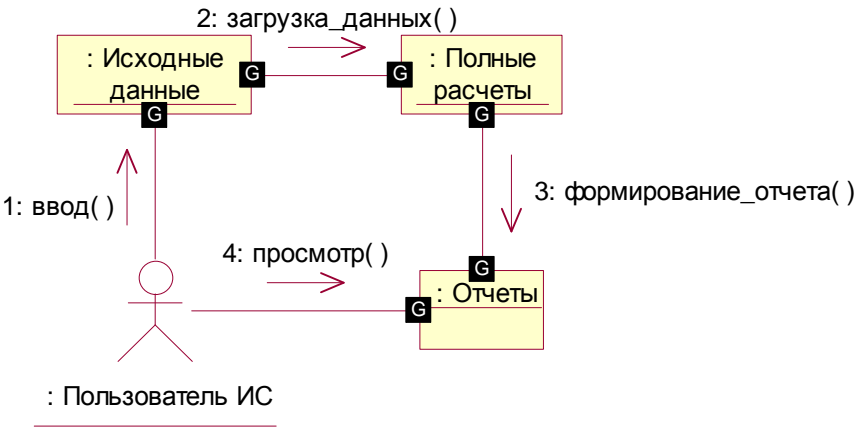


Рисунок 160 – Диаграмма кооперации (полный расчет)

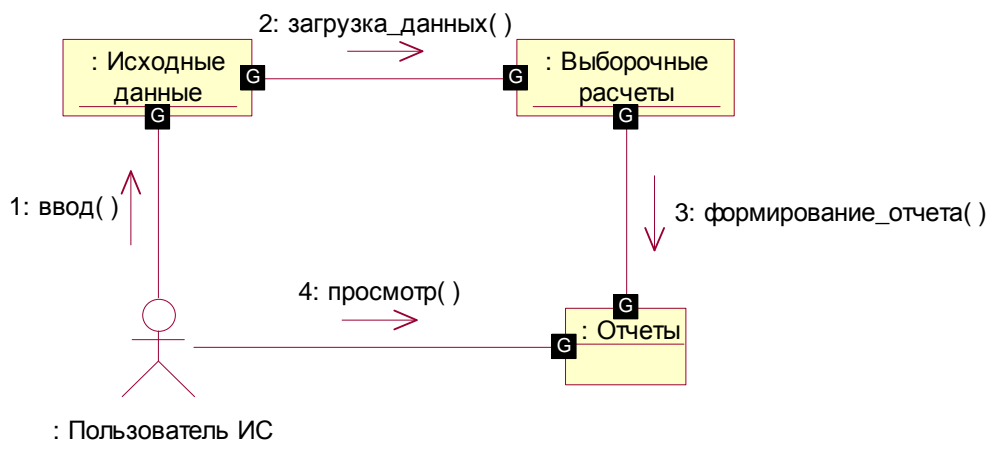


Рисунок 161 – Диаграмма кооперации (выборочный расчет)

Все описанные диаграммы описывают лишь некую абстракцию, существующую лишь виртуально. Для описания более реальной и доступной для понимания физической формы системы применяются диаграммы реализации – диаграмма компонентов и диаграмма развертывания. Диаграмму компонентов проектируемой системы можно рассматривать как диаграмму проекта системы (рис.162) и диаграмму компонентов реализованной системы (рис. 163).

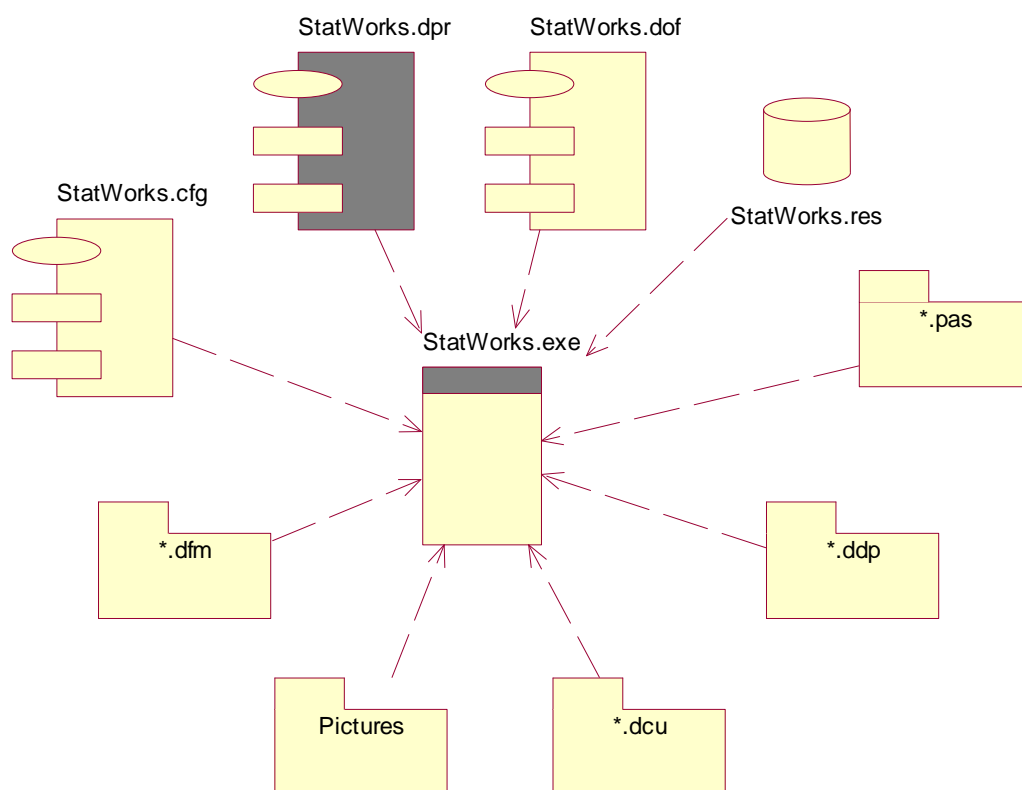


Рисунок 162 – Диаграмма компонентов проекта системы

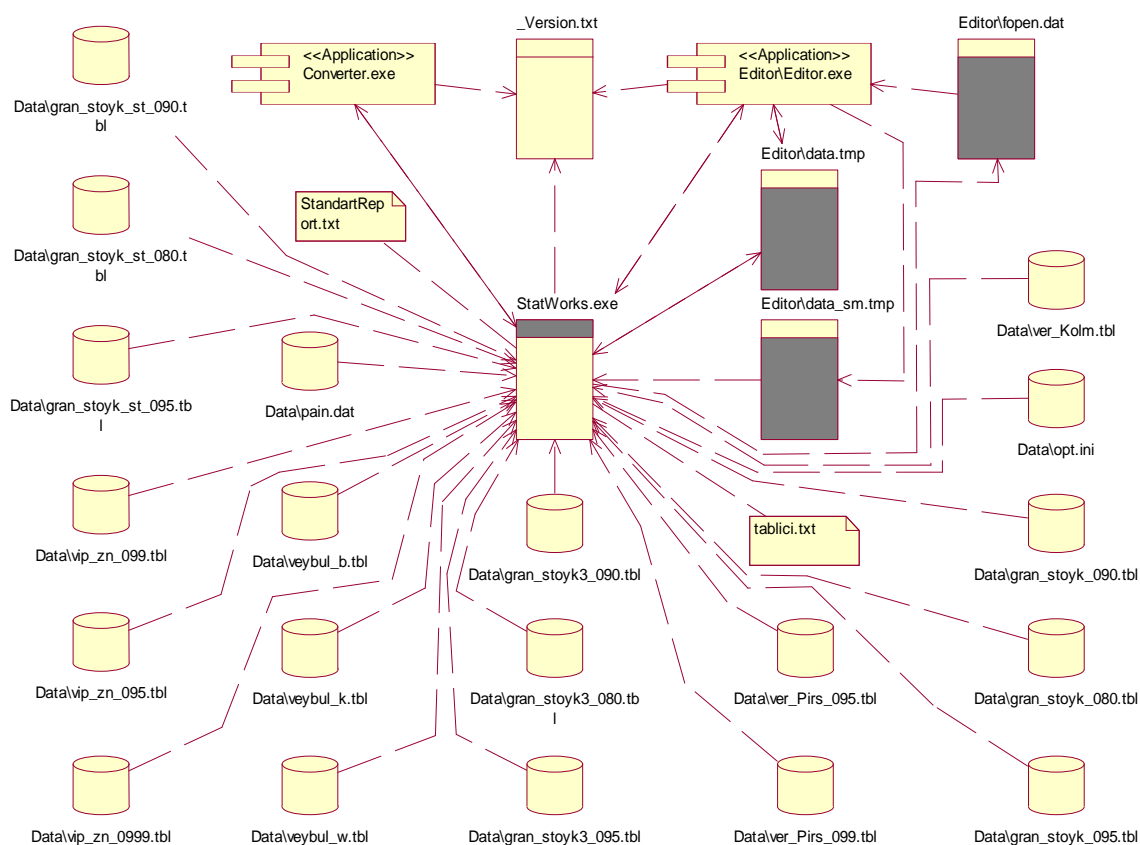


Рисунок 163 – Диаграмма компонентов реализованной системы

Поскольку проектируемая информационная система не предполагает сетевую работу, то диаграмма развертывания не строится.

Программная реализация системы была выполнена в среде Borland Delphi 6.0. Проект имел название StatWorks и включал в себя 12 форм интерфейса и более 6,5 тыс. строк исходного текста [29]. В настоящее время программа используется в учебном процессе кафедры «Металло-режущие станки и инструменты» ДГМА.

СПИСОК ЛІТЕРАТУРИ

1. Грейди Буч. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. – 2-е изд. / Пер. с англ. – М.: Издательство Бином; СПб.: Невский диалект, 2001. – 560 с.
2. Леоненков А.В. Самоучитель UML. – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2004. – 432 с.
3. Грейди Буч. Язык UML. Руководство пользователя / Грейди Буч, Джеймс Рамбо, Айвар Джекобсон; Пер. с англ. А.А.Слинкин. – 2-е изд., стер. – М.: ДМК Пресс; СПб.: Питер, 2004. – 432 с.
4. Т. Кватрани. Rational Rose 2000 и UML. Визуальное моделирование. – СПб.: ДМК-Пресс, 2000. – 176 с.
5. UML и Rational Rose / Под ред. У.Боггс, М.Боггс. – М.: Лори, 2001. – 608с.
6. UML / Под ред. М. Фауллера, К. Скотта. – М.: Мир, 1999. – 191с.
7. Шмулер Джозеф. Освой самостоятельно UML за 24 часа. – 2-е издание/ Пер. с англ. – М.: Издательский дом «Вильямс», 2002. – 352 с.
8. Бабич А.В. Обзор CASE-средств для построения диаграмм UML // Теорія та методика навчання математики, фізики, інформатики: Збірник наукових праць. Випуск 5: В 3-х томах. – Кривий Ріг: Видавничий відділ НМетАУ, 2005. – Т.3: Теорія та методика навчання інформатики. – С. 8-11.
9. Васись Д.В. Информационная система для функционирования кадрового агентства/ Д.В.Васись, А.Ю.Мельников // Сборник докладов 2-го молодежного форума «Информационные технологии в XXI-м веке» (27-28 апреля 2004 г., г. Днепропетровск). – Днепропетровск: ИПК Ин-КомЦентра УГХТУ, 2004. – С.31-32.
10. Васись Д.В. Інформаційна система для функціонування кадрового агентства/ Д.В.Васись, О.Ю.Мельников // Інформаційні технології в освіті, науці і техніці: Матеріали IV Всеукраїнської конференції молодих науковців ІТОНТ-2004: Черкаси, 28-30 квітня 2004 р. – Черкаси: ЧНУ, 2004. – С.87-89.

11. Мельников А.Ю. Информатизация функционирования кадрового агентства/ А.Ю.Мельников, Д.В.Васись// Вісн. Східноукр. нац. ун-ту ім. В. Даля – Луганськ. – 2004. – №11. – С. 230-234.

12. Мельников А.Ю. Проектирование информационной системы для функционирования кадрового агентства/ А.Ю.Мельников, Д.В.Васись // Современные проблемы информатизации в непроизводственной сфере и экономике: Сб. трудов. – Вып. 10 / Под ред. д.т.н., проф. О.Я. Кравца. – Воронеж: Издательство «Научная книга», 2005. – С.14-15.

13. Мельников А.Ю. Объектно-ориентированное проектирование информационной системы для функционирования кадрового агентства/ А.Ю.Мельников, Д.В.Васись // Вісник Донбаської державної машинобудівної академії: Збірник наукових праць. – №1. – 2005. – Краматорськ: ДДМА, 2005. – С. 233-237.

14. Мельников А.Ю. Автоматизация процесса составления расписания занятий в высшем учебном заведении/ А.Ю.Мельников, Н.М.Сусяк // Комп'ютерне моделювання в освіті: Матеріали Всеукраїнського науково-методичного семінару. – 29 березня 2005р. – Кривий Ріг: КДПУ, 2005. – С.52-53.

15. Сусяк Н.М. Проектирование системы для автоматизации составления расписания занятий в вузе/ Н.М.Сусяк, А.Ю.Мельников // Информационные технологии в XXI веке: Сборник докладов и тезисов III-го Молодежного научно-практического форума (Днепропетровск, 27-28 апреля 2005г.) / Под ред. акад. НАНУ В.В. Пилипенко, д.х.н. М.В. Бурмистра, к.ф.-м.н. Н.Ф. Огданского, к.ф.-м.н. Ю.А. Прокопчука. – Днепропетровск: ИПК ИнКомЦентра УГХТУ, 2005. – С. 189-190.

16. Мельников О.Ю. Проектування системи для автоматизованого складання розкладу занять у вищому навчальному закладі/ О.Ю.Мельников, Н.М.Сусяк // Проблеми прийняття рішень в умовах невизначеності: Матеріали міжнародної науково-практичної конференції (м. Бердянськ, 12-17 вересня 2005 року). – Бердянськ, 2005. – С. 70-71.

17. Мельников О.Ю. Проектування системи для автоматизованого складання розкладу занять у вищому навчальному закладі / О.Ю.Мельников, Н.М.Сусяк // Збірник наукових праць Бердянського

державного педагогічного університету (Педагогічні науки). – № 3. – Бердянськ: БДПУ, 2005. – С. 40-46.

18. Мельников А.Ю. Автоматизированное составление расписания занятий в высшем учебном заведении/ А.Ю.Мельников, Н.М.Сусяк // Современные проблемы информатизации в информационных системах и телекоммуникациях: Сб. трудов. – Вып. 11 / Под ред. д.т.н., проф. О.Я.Кравца. – Воронеж: Издательство «Научная книга», 2006. – С.344-345.

19. Мельников А.Ю. Система для автоматизированного составления расписания занятий в высшем учебном заведении/ А.Ю.Мельников, Н.М.Сусяк // Открытое и дистанционное образование. – Томск, 2006. – № 2 (22). – С. 52-57.

20. Мельников А.Ю. Разработка информационной системы для специализированного торгового предприятия/ А.Ю.Мельников, А.В.Руденко // Восточно-Европейский журнал передовых технологий. – Харьков, 2005. – №4/2(16). – С. 124-127.

21. Мельников А.Ю. Проектирование информационной системы для специализированного торгового предприятия/ А.Ю.Мельников, А.В.Руденко // Современные проблемы информатизации в информационных системах и телекоммуникациях: Сб. трудов. – Вып. 11 / Под ред. д.т.н., проф. О.Я. Кравца. – Воронеж: Издательство «Научная книга», 2006. – С.37-38.

22. Ольховська О.Л. Проектування інформаційної системи функціонування страхової компанії/ О.Л.Ольховська, О.Ю.Мельников // Информационные технологии в XXI веке: Сборник докладов и тезисов III-го Молодежного научно-практического форума (Днепропетровск, 27-28 апреля 2005г.) / Под ред. акад. НАНУ В.В. Пилипенко, д.х.н. М.В. Бурмистра, к.ф.-м.н. Н.Ф. Огданского, к.ф.-м.н. Ю.А. Прокопчука. – Днепропетровск: ИПК ИнКомЦентра УГХТУ, 2005. – С. 147-148.

23. Мельников А.Ю. Проектирование информационной системы для функционирования страховой компании/ А.Ю.Мельников, О.Л.Ольховская // Современные проблемы информатизации в информационных системах и телекоммуникациях: Сб. трудов. – Вып. 11 / Под

ред. д.т.н., проф. О.Я. Кравца. – Воронеж: Издательство «Научная книга», 2006. – С.38-39.

24. Мельников А.Ю. Разработка информационной системы функционирования страховой компании/ А.Ю.Мельников, О.Л.Ольховская // Информационные технологии моделирования и управления: Научно-технический журнал. – Воронеж: Издательство «Научная книга», 2006. – №2 (27). – С.277-283.

25. Мельников О.Ю. Проектування інформаційної системи для невеликої страхової компанії/ О.Ю.Мельников, О.Л.Ольховська // Вісник Національного технічного університету «Харківський політехнічний інститут»: Збірник наукових праць. Тематичний випуск: Інформатика і моделювання. – Харків: НТУ «ХПІ». – 2005. – № 56. – С. 91-99.

26. Мельников А.Ю. Методы расчета себестоимости металлопродукции/ А.Ю.Мельников, В.Ю.Гуржиев // Сборник тезисов IV Международной научно-практической конференции молодых ученых и специалистов «Интеллект молодых – производству 2005». – Краматорск, 2005. – С. 130-131.

27. Мельников А.Ю. Автоматизация расчета себестоимости/ А.Ю.Мельников, В.Ю.Гуржиев // Важке машинобудування. Проблеми та перспективи розвитку. Матеріали четвертої Міжнародної науково-технічної конференції 5-8 червня 2006 року / Під заг.ред. В.Д.Ковальова. – Краматорськ: ДДМА, 2006. – С.68.

28. Ключков Е.В. Автоматизация учета и контроля производственной деятельности КИГАЗ «Авиант»: учет и контроль готовой продукции // Збірник (частина 2) тез доповідей IX Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених “Технологія-2006” (13-14 квітня 2005 р., м.Северодонецк) / СТІ СХУ ім. В.Даля. – Северодонецьк, 2006. – С. 59.

29. Мельников А.Ю. Разработка информационной системы для маркетинговых исследований и анализа надежности/ А.Ю.Мельников, В.Л.Аносов, Д.Е.Прекрасный // Восточно-Европейский журнал передовых технологий. – Харьков, 2005. – №1 (19). – С. 122-127.

Навчальне видання

МЕЛЬНИКОВ Олександр Юрійович

**ОБ’ЄКТНО-ОРІЄНТОВАНИЙ АНАЛІЗ
І ПРОЕКТУВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ**

Навчальний посібник

(Російською мовою)

Редактор

О.О.Дудченко

Підп. до друку **22.11.06** Формат 60х84/16
Папір офсетний. Ризограф. друк. Ум. друк.арк **11,5**. Обл.-вид.арк. **8,36**.
Тираж **150** прим. Зам.№ **301**

Видавець і виготівник
«Донбаська державна машинобудівна академія»
84313, м. Краматорськ, вул. Шкадінова, 72
Свідоцтво про внесення суб’єкта видавничої справи
до Державного реєстру
серія ДК №1633 від 24.12.2003 р.