МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра вычислительной техники

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г. Локтионова
«_15_» _01_ 2021 г.

УПРАВЛЕНИЕ ПРОЕКТИРОВАНИЕМ ИНФОРМАЦИОННЫХ СИСТЕМ

Методические указания к выполнению лабораторных работ по дисциплине «Управление проектированием информационных систем» для студентов направления подготовки 09.04.01

Курск 2021

УДК 004

Составитель: О.О. Яночкина

Рецензент
Кандидат технических наук, доцент *Ю.А. Халин*

Управление проектированием информационных систем: методические указания к выполнению лабораторных работ по дисциплине «Управление проектированием информационных систем» / Юго-Зап. гос. ун-т; сост.: О.О. Яночкина, Курск, 2021. 44 с.: Библиогр.: с. 44.

Методические указания содержат сведения о порядке выполнения лабораторных работ, характеристику ожидаемых результатов и требования к оформлению отчета.

Методические указания соответствуют Федеральному государственному образовательному стандарту высшего образования - магистратура по направлению подготовки 09.04.01 Информатика и вычислительная техника, утвержденному приказом № 918 от 19.09.2017

Предназначены для студентов направления подготовки 09.04.01 очной формы обучения.

Текст печатается в авторской редакции

Подписано в печать 15.09 . Форма 60x84 1/16.
Усл. печ. л. . Уч.-изд.л. . Тираж ___ экз. Заказ. 232
Бесплатно
Юго-Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

ЛАБОРАТОРНАЯ РАБОТА №1

“Исследование предметной области. Диаграммы прецедентов и диаграммы действий"

1. Цель работы

Целью работы является исследование предметной области, изучение процесса построения диаграмм прецедентов и диаграммы действий с помощью инструмента объектного моделирования Rational Rose.

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1. Диаграммы прецедентов

В соответствии с методологией объектно-ориентированного анализа и проектирования первым этапом является анализ требований, который подразумевает выделение процессов и требований и их формулировку в виде прецедентов.

Прецедент в объектном моделировании (англ. – use case) представляет собой документ, описывающий последовательность событий, связанных с исполнителем (внешним агентом), который для завершения требуемого процесса использует создаваемую систему. Прецеденты являются описанием или вариантами использования системы. С помощью прецедента описывается некоторый процесс.

По результатам анализа прецедентов на первом этапе моделирования предметной области создается диаграмма определения требований к системе Use Case (сценарии поведения). Данная диаграмма позволяет создавать диаграммы поведения объектов системы.

На диаграмме прецедентов иллюстрируется набор прецедентов системы и исполнители, а также взаимосвязи между ними. Прецеденты определяют, как исполнители взаимодействуют с программной системой. В процессе этого взаимодействия исполнителем генерируются события, передаваемые системе, которые представляют собой запросы на выполнение некоторой операции.

Диаграмма прецедентов содержит:

- варианты использования (прецеденты) системы (use case);
- действующее лицо (actor).

Диаграмма отражает взаимодействие вариантов использования и действующих лиц. Она отражает требования к системе с точки зрения пользователя.

Варианты использования системы – описание функций системы на "высоком уровне". Они описывают все, что происходит внутри области действия системы. Варианты использования иллюстрируют, как можно использовать систему. Они заостряют внимание на том, что пользователи хотят получить от системы. Каждый вариант использования представляет собой завершенную транзакцию между пользователем и системой.

Действующее лицо – все, что взаимодействует с системой, передает или получает информацию от системы. Исполнитель (actor) является

внешним по отношению к системе понятием, которое определенным образом участвует в процессе, описываемом прецедентом. Они описывают все, что находится вне системы. Это пользователи системы, другие системы, взаимодействующие с описываемой, время.

Каждый прецедент должен быть инициирован действующим лицом.

В качестве примера рассмотрим процесс настройки и внедрения программных продуктов.

Работы по настройке и внедрению распределяются на следующие этапы:

- обследование объекта;
- организационные мероприятия;
- разработка технического задания по результатам обследования;
- адаптация и/или доработка программ к специфике предприятия (организации);
- внедрение программного продукта.

Заказчиком формулируются требования к информационной системе, разработчик изучает автоматизируемый процесс, при этом выявляет основные характеристики будущей системы – составляет спецификации.

Далее разработчик выбирает программный продукт, удовлетворяющий спецификациям системы. Выбор согласовывается с заказчиком по стоимости программы и другим характеристикам. Разрабатывается техническое задание на настройку и внедрение выбранного ПО к объекту автоматизации. Далее идет этап непосредственной настройки программного продукта (адаптация и/или доработка). При этом возможно как использование существующих модулей, так и создание новых.

Далее идут этапы внедрения, эксплуатации программного продукта.

При моделировании процесса сначала проводится описание системы в целом и ее взаимодействия с окружающим миром – создается диаграмма прецедентов или вариантов использования (Use Case Diagram).

При создании диаграмм прецедентов вначале определяются исполнители (роли, пользователи). В нашем примере исполнителями являются "заказчик" и "разработчик".

Исполнитель может быть абстрактным, не имеющим экземпляров. В нашем случае, абстрактным исполнителем выступает "разработчик". Его разновидностями являются "начальник отдела" и "программист". Абстрактный исполнитель существует для того, чтобы показать общность между этими типами.

Следующий шаг - идентификация прецедентов. У каждого прецедента должно быть уникальное имя.

В рассматриваемом примере прецедентами будут этапы работ по настройке и внедрению.

Последний шаг – определяются и проставляются связи.

В диаграммах прецедентов поддерживается несколько типов связей:

- связи коммуникации (Communication) – описывают связи между действующими лицами и вариантами использования;
- использования (uses) и расширения (extends) – отражают связи между вариантами использования;
- обобщения действующего лица (actor generalization) – между

действующими лицами. В примере это связь между разработчиком и программистом, разработчиком и начальником отдела.

Связь использования позволяет одному варианту использования задействовать функциональность другого. С помощью таких связей обычно моделируют многократно применяемую функциональность, встречающуюся в двух или более вариантах использования. Например, прецедент "Изучить объект автоматизации" задействует функции прецедента "Составить спецификации".

Связь расширения позволяет варианту использования только при необходимости применять функциональные возможности другого варианта (extends). Например, прецедент "Настроить программный продукт" при необходимости использует функции прецедента "написать новые программные модули" или "использовать готовые программные модули".

Диаграмма процесса настройки и внедрения программных продуктов представлена на рис.3.

Часто, чтобы избежать большого количества прецедентов на диаграмме, создают вложенные диаграммы. В нашем примере это диаграмма "Исследовать объект автоматизации" (рис.4).

Разрабатывая диаграммы, придерживаются правил:

1. Не моделируют связи между действующими лицами. По определению они находятся вне сферы действия системы. Связи между ними не относятся к ее компетенции.

2. Не соединяют стрелкой непосредственно два варианта использования (кроме связей использования и расширения). Диаграмма описывает только, какие варианты использования доступны системе, а не порядок их выполнения.

3. Каждый вариант использования должен быть инициирован действующим лицом. Всегда должна быть стрелка, начинающаяся на действующем лице и заканчивающаяся на варианте использования (кроме связей использования и расширения).

4. Думают о БД, как о слое, находящемся под диаграммой. С помощью одного варианта использования можно вводить данные в базу, а получать их – с помощью другого. Не рисуют стрелки от одного варианта к другому для изображения потока информации.

Конкретные детали вариантов использования отражаются в основном и альтернативном потоках событий. Поток событий поэтапно описывает, что должно происходить во время выполнения заложенной в варианты использования функциональности. Поток событий уделяет внимание тому, что (а не как) будет делать система, причем описывает это с точки зрения пользователя. Первичный и альтернативный потоки событий содержат:

– описание того, каким образом запускается вариант использования,

– различные пути выполнения варианта использования,

– нормальный, или основной, поток событий варианта использования,

– отклонения от основного потока событий (так называемые альтернативные потоки),

– потоки ошибок,

– описание того, каким образом завершается вариант

использования.

Рис.3 - Диаграмма процесса настройки и внедрения программных продуктов

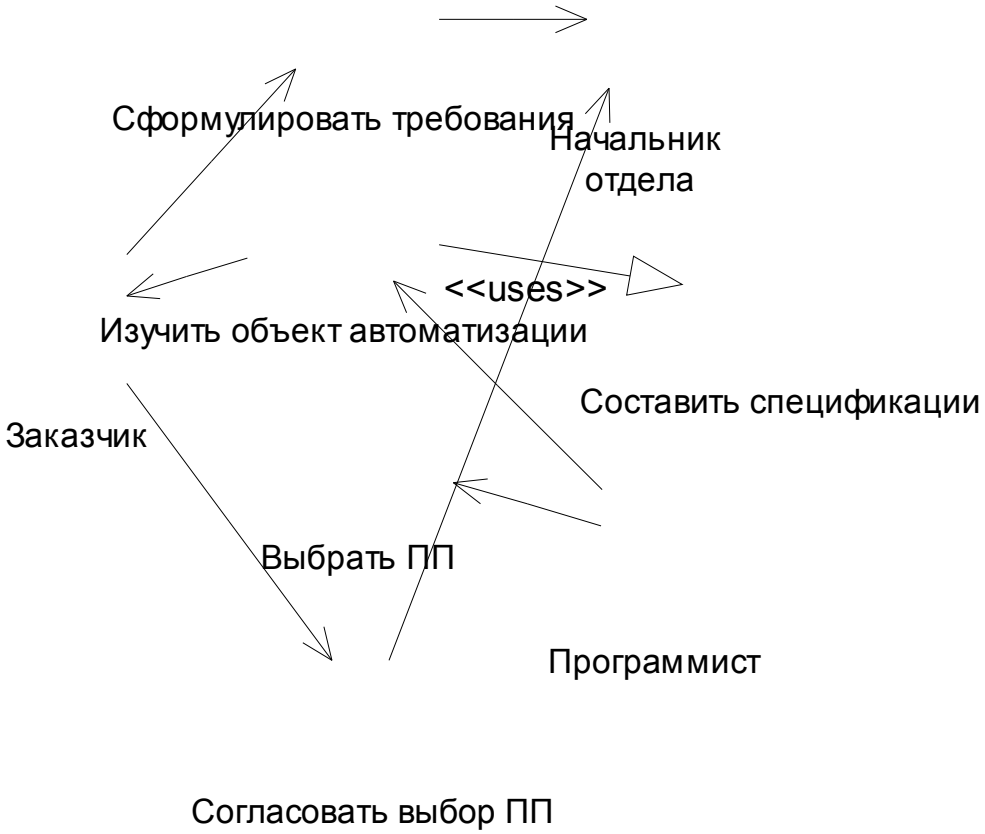

Рис.4 - Диаграмма прецедентов "Исследовать объект автоматизации"

Можно создать подробную спецификацию для каждого варианта использования. Они помогают документировать такие атрибуты вариантов использования, как имена, приоритеты и стереотипы. Можно размещать на диаграмме описания (Note).

На основе набора Use Case диаграмм создается список требований к системе и определяется множество выполняемых системой функций.

2.2. Диаграммы действий

При моделировании поведения проектируемой системы возникает необходимость не только представить процесс изменения её состояний, но и детализировать особенности алгоритмической и логической операции выполняемой системой реализации. Традиционно для этой цели использовались блок-схема или структурные схемы алгоритмов. В UML для этого используется диаграмма действий.

На диаграмме деятельности отображается логика или последовательность переходов от одной деятельности к другой. При этом внимание фиксируется на результате деятельности. Сам же результат может привести к изменению состояния системы или возвращению некоторого значения.

Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого является состояние действия, а дугами – переходы от одного состояния в действия к другому.

При этом каждое состояние может являться выполнением операции некоторого класса либо её часть, позволяя использовать диаграмму деятельности для описания реакции на внутренние события системы.

На рис.5 представлен алгоритм процесса принятия решения при настройке и внедрении программного продукта, описанный с помощью диаграммы действий.

В языке UML действие изображается в виде прямоугольника с закругленными углами, состояния - в виде прямоугольника, переходы – в виде направленных стрелок, элементы выбора – в виде ромбов, линии синхронизации – в виде толстых горизонтальных или вертикальных линий.

Состоянием называется одно из возможных условий, в которых может существовать объект. Для выявления состояний объекта необходимо исследовать две области модели: значения атрибутов объекта и связи с другими объектами. В рассматриваемом примере: "Сформированы требования к ПП", "Выданы рекомендации".

Существуют специальные состояния объекта: начальное и конечное. Начальным (Start State) называется состояние, в котором находится объект сразу после своего создания (В примере это "заказ на ПО"). Конечным (End State) называется состояние, в котором объект находится непосредственно перед уничтожением ("Решение по адаптации").

Действием называется исполнение определенного поведения в потоке управления системы. В примере это "Поиск правила", "Формировать решение" и т.д.

Переходы используются для изображения пути потока управления от действия к действию.

Элементы выбора показывают места разделения управляющих потоков на основе условного выбора. Переходы из элементов выбора содержат ограничительные условия, определяющие, какое направление перехода будет выбрано ("Поиск правила", "Поиск прецедента").

Линии синхронизации (synhronization bar) позволяет указать на необходимость одновременного выполнения действий, а также обеспечивает единое выполнение действий в потоке ("Формировать решение", "Адаптировать решение").

Секции делят диаграмму действий на несколько участков, чтобы показать, кто отвечает за выполнение действий на каждом участке.

Диаграммы действий отражают динамику проекта и представляют собой схемы потоков управления в системе от действия к действию, а также параллельные действия и альтернативные потоки.

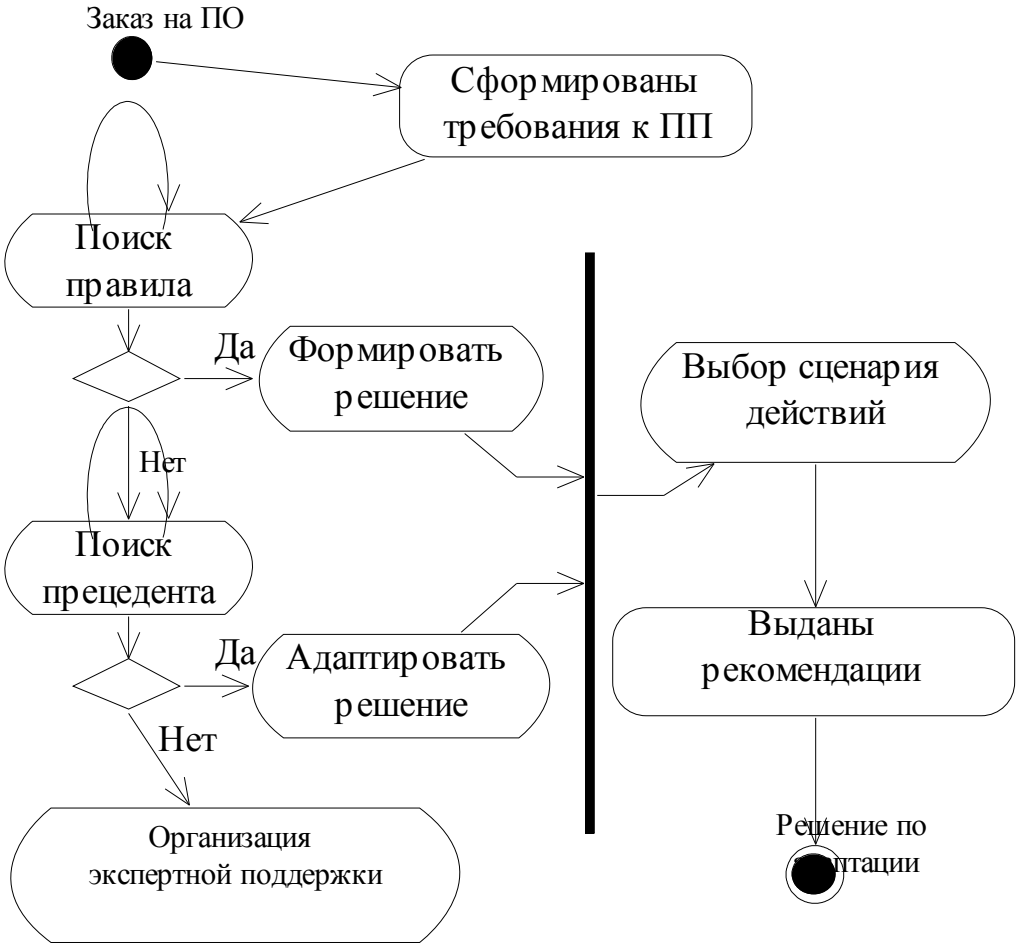

Рис. 5 - Диаграмма действий

2.3. Главное окно "Rational Rose"

В верхней части экрана, как у большинства редакторов в стиле Windows, находится меню и панель инструментов (Tool Bar).

Слева находится окно Browser для быстрого доступа к диаграммам. Это окно позволяет легко перемещаться по дереву диаграмм, буксировать диаграммы мышкой и изменять структуру модели по своему усмотрению.

Под окном Browser находится окно Documentation. В этом окне появляется описание, которое введено разработчиком для выделенного в текущий момент элемента.

В правой части экрана находятся те диаграммы, которые открыты в текущий момент, обычно это поле называется Рабочим столом Rational Rose.

Между окном Browser и окном Diagram находится панель инструментов текущей диаграммы, которая изменяется в зависимости от выбранной диаграммы.

Внизу рабочего стола находится свернутое окно Log (протокол). В

нем постоянно фиксируются все действия, произведенные над диаграммами.

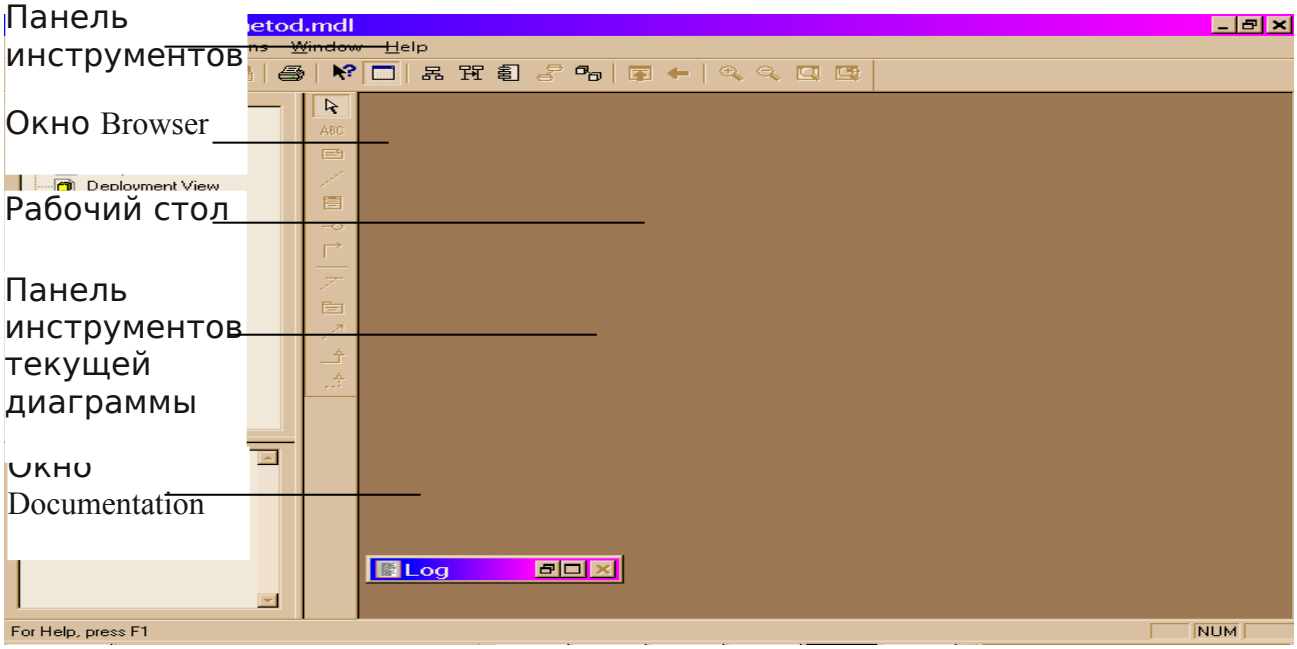

Рис.6 - Главное окно Rational Rose

3. ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

3.1. Построение диаграммы прецедентов

1. Откройте Главную диаграмму прецедентов (окно Browser > Use Case View >Main) или создайте новую диаграмму (.Use Case View >New>Use Case Diagram). (рис.7)

2. Создайте исполнителей, поместив их непосредственно на рабочий стол Rational Rose из строки инструментов текущей диаграммы, или выполнив последовательность: Use Case View>New>Actor. Атрибуты и операции можно задать из контекстного меню на рабочем столе или в окне броузера (New Attribute и New Operation). (рис.8)

3. Создайте прецеденты, поместив их непосредственно на

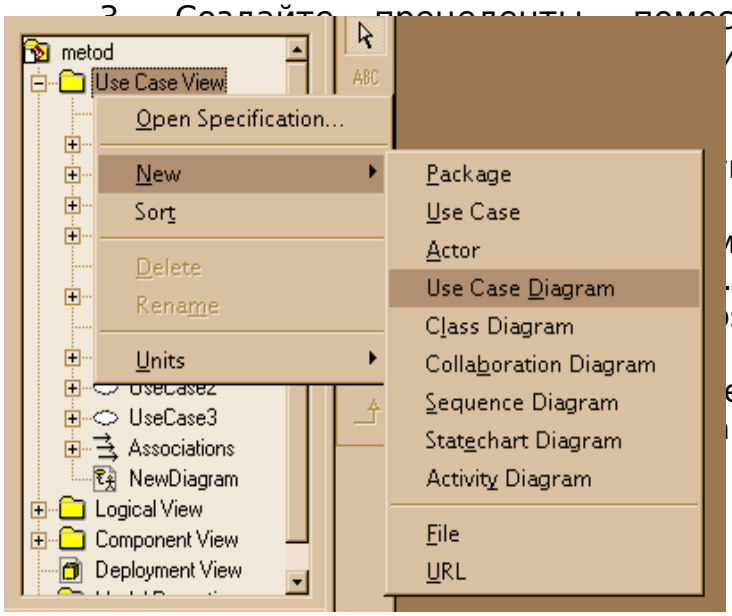

Рис.7

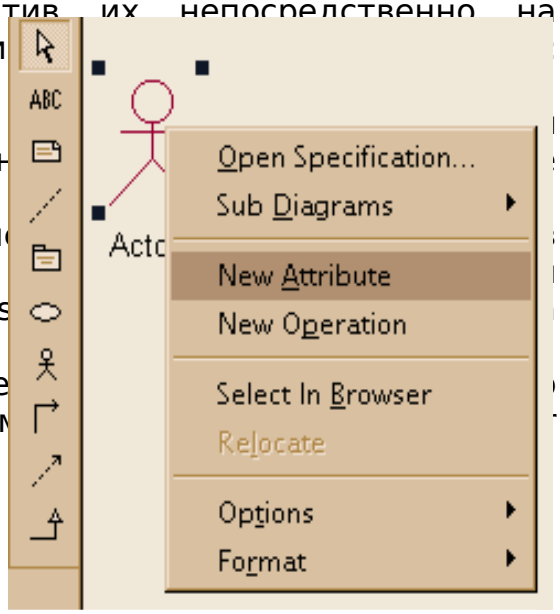

Рис.8

3.2. Построение диаграммы действий

1. Создайте диаграмму действий (.New> Activity Diagram).
2. Создайте необходимые элементы (состояния, действия, условия, связи) поместив их на рабочий стол из строки инструментов. Не забудьте про начальные и конечные состояния.

3.3. Порядок выполнения лабораторной работы

1. Выберите свой Вариант задания и предметную область для моделирования из файла «Варианты практических заданий по МИиМИПиС.doc».
2. Исследуйте систему.
3. Постройте диаграмму прецедентов или, если требуется, несколько диаграмм.
4. Постройте диаграмму действий для описания какого-либо процесса.
5. Ответьте на контрольные вопросы.

4. Контрольные вопросы

1. В чем состоит сущность объектно-ориентированного подхода?
2. Назовите основные понятия объектно-ориентированного подхода.
3. Что понимается под объектом и классом в объектном моделировании?
4. Назовите главные принципы объектно–ориентированного анализа.
5. Что такое диаграмма в UML?Типы диаграмм.

6. Назовите типы сущностей и связей в UML.
7. Что понимается под прецедентом в объектном моделировании?
8. Для чего строится диаграмма прецедентов?
9. Перечислите основные элементы диаграммы прецедентов.
10. Какие типы связей поддерживаются в диаграммах прецедентов?
11. Для чего строятся диаграммы действия?
12. Перечислите основные элементы диаграммы действий.

ЛАБОРАТОРНАЯ РАБОТА №2

“Разработка, моделирование и анализ структуры информационной системы. Диаграммы классов"

1. Цель работы

Целью работы является разработка, моделирование, анализ структуры информационной системы и исследование процесса построения диаграммы классов в заданной предметной области с помощью пакета Rational Rose.

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

На диаграммах классов отображаются некоторые классы и пакеты системы. Это статические картины фрагментов системы и связей между ними.

Обычно для описания системы создают несколько диаграмм классов. На одних показывают некоторое подмножество классов и отношения между классами подмножества. На других отображают то же подмножество, но вместе с атрибутами и операциями классов. Третьи соответствуют только пакетам классов и отношениям между ними. Для представления полной картины системы можно разработать столько диаграмм классов, сколько требуется.

2.1. Диаграмма пакетов

Пакеты (packages) применяются для группирования классов, обладающих некоторой общностью.

По умолчанию существует одна диаграмма классов, называемая Главной (Main). На этой диаграмме показывают пакеты классов модели. Внутри каждого пакета также имеется Главная диаграмма, включающая в себя все классы этого пакета.

В качестве примера рассмотрим главную диаграмму классов, описывающую логическое представление системы поддержки принятия решений при адаптации и внедрении программных продуктов. СППР помогает разработчику при выборе программного продукта, выполняя функции эксперта.

Работая с базой знаний прецедентов, пользователь вводит простой запрос, содержащий набор признаков, описывающих объект автоматизации – спецификации заказа на программный продукт. Решение ищется сначала в базе правил, затем, если соответствующее правило не сформулировано, в базе прецедентов.

Пакет классов «Заказ» представляет собой заказ на программный продукт с описанием требований к системе (рис.9). Пакет классов «База прецедентов» представляет собой описание структуры базы знаний прецедентов – описания существующих программных продуктов. Прецедент – это описание программного продукта в совокупности с подробным сценарием действий, предпринимаемых в данной ситуации для адаптации данного программного продукта. Пакет классов «Действия по адаптации» описывает структуру базу правил по адаптации

программных продуктов – рекомендации для разработчиков.

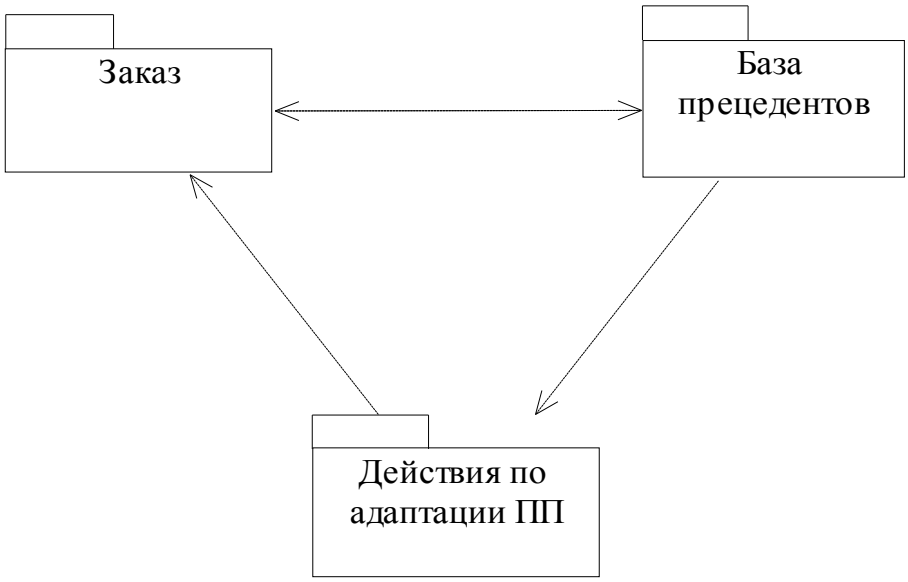

Рис.9 - Диаграмма пакетов: СППР при адаптации и внедрении программных продуктов

2.2. Диаграммы классов

Для задания класса необходимо указать имя этого класса, а затем перечислить его атрибуты и операции (или методы). Полное описание объекта на графическом языке ОМТ имеет вид, изображенный на рисунке 10.

ИМЯ_КЛАССА
имя_атрибута 1: тип_1 = *значение_по_умолчанию_1* *имя_атрибута 2: тип_2 =* *значение_по_умолчанию_2* *. . . . . .*
имя_операции_1 (список_аргументов_1): *тип_результата* *сигнатура_операции_2* *сигнатура_операции_3* *. . . . . .*

Рис. 10. Полное представление объекта в ОМТ

Атрибут - это значение, характеризующее объект в его классе. Примеры атрибутов: имя признака, тип признака, значение признака, вес признака (атрибуты объектов класса "Признак") и т.д.

Среди атрибутов различаются постоянные атрибуты (константы) и переменные атрибуты. Постоянные атрибуты характеризуют объект в его классе (например, имя человека, имя признака и т.п.). Текущие значения переменных атрибутов характеризуют текущее состояние объекта

(например, возраст человека, значение признака и т.п.); изменяя значения этих атрибутов, мы изменяем состояние объекта.

Иногда указывается тип атрибутов (ведь каждый атрибут - это некоторое значение) и начальное значение переменных атрибутов (совокупность начальных значений этих атрибутов задает начальное состояние объекта).

Операция - это функция (или преобразование), которую можно применять к объектам данного класса. Одна и та же операция может, вообще говоря, применяться к объектам разных классов: такая операция называется полиморфной, так как она может иметь разные формы для разных классов.

Каждой операции соответствует метод - реализация этой операции для объектов данного класса. Таким образом, операция - это спецификация метода, метод - реализация операции. Например, в классе файл может быть определена операция печать (print). Эта операция может быть реализована разными методами: (а) печать двоичного файла; (б) печать текстового файла и др. Логически эти методы выполняют одну и ту же операцию, хотя реализуются они разными фрагментами кода.

Каждая операция имеет один неявный аргумент - объект к которому она применяется. Кроме того, операция может иметь и другие аргументы (параметры). Эти дополнительные аргументы параметризуют операцию, но не связаны с выбором метода.

Операция (и реализующие ее методы) определяется своей сигнатурой, которая включает, помимо имени операции, типы (классы) всех ее аргументов и тип (класс) результата (возвращаемого значения). Все методы, реализующие операцию должны иметь такую же сигнатуру, что и реализуемая ими операция.

На рис.11 представлена диаграмма классов для описания структуры базы знаний прецедентов описаний программных продуктов. В базе знаний прецеденты по общим признакам объединены в классы, так называемые, прецеденты–концепты. Классы "Прецедент" и "Прецедент-концепт" соединены связью агрегации (связь между целым и его частями). У прецедентов (прецедентов-концептов) есть атрибут "Признак", который в то же время является классом как сложный объект, имеющий свои атрибуты. "Тип признака" тоже определен как класс. Для каждого признака существует процедура сравнения – класс "Сравнение". Класс «Решение» - решения по адаптации и внедрению для конкретных прецедентов и классов прецедентов.

Каждому классу модели Rose дают уникальное имя. В общем случае используются существительные в единственном числе.

Обычно имена классов не содержат пробелов. Это делается по практическим причинам и из соображений удобочитаемости — языки программирования, как правило, не поддерживают пробелы именах классов.

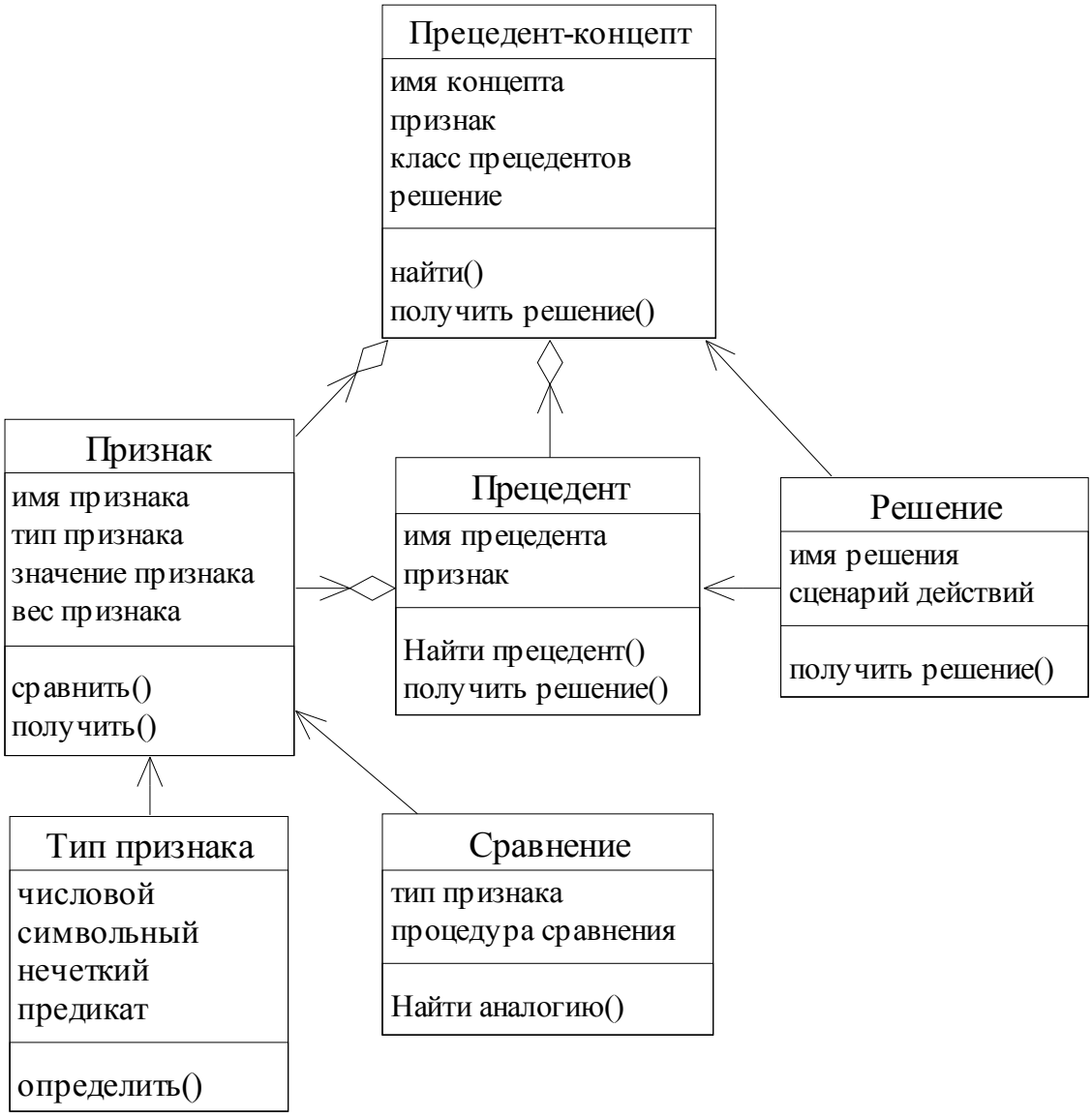

Рис.11 - Диаграмма классов: База прецедентов

Параметр Visibility (Видимость) показывает, будет ли класс виден вне своего пакета:

1. Public (открытый) - этот класс виден всем остальным классам системы.
2. Protected, Private (защищенный, закрытый) - класс может быть виден во вложенных в него классах, "друзьям" (friends) этого класса или из самого класса.
3. Package or Implementation (пакет или реализация) - класс может быть виден только из классов того же пакета.

Поле Cardinality (Множественность) позволяет указать, сколько у данного класса должно быть экземпляров.

Абстрактным называется класс, который не наполняется конкретным содержанием. Иными словами, если класс А абстрактный, в памяти никогда не будет объектов типа А.

Обычно абстрактные классы применяют при работе с наследованием. В них содержатся данные и поведение, общие для нескольких других классов.

В Rose классы можно вкладывать друг в друга. Во вложенные (nested) классы можно вкладывать другие классы, организуя столько уровней вложения, сколько необходимо.

Поведение системы обеспечивается взаимодействием объектов. Два типа отношений, которые можно выделить на этапе анализа – это ассоциация и агрегация.

Ассоциация (association) – это двунаправленная семантическая связь между классами.

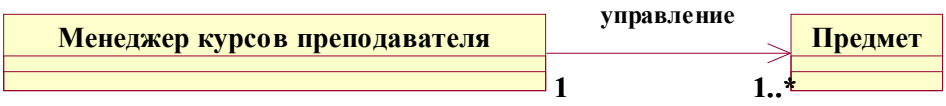

Рис.12

Агрегационное отношение – это специальная форма ассоциации между целым и его частью или частями, то есть отношение типа «часть от» или «содержит».

Рис.13

Мощность отношения (multiplicity) указывается для классов и определяет допустимое количество объектов, участвующих в отношении с каждой стороны.

Несколько объектов, принадлежащих одному классу, могут взаимодействовать друг с другом. Такое взаимодействие показывается на диаграмме классов как возвратное.

Наследованием называется такое отношение между классами, когда один класс использует часть структуры и / или поведения другого или нескольких классов. При наследовании создается иерархия абстракций, в которой подкласс (subclass) наследуется от одного или нескольких суперклассов (superclass). Подкласс наследует все атрибуты, операции и отношения, определенные в каждом его суперклассе.

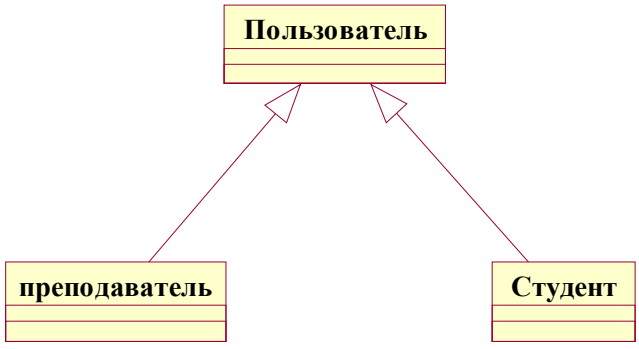

Рис.14

Диаграммы Классов являются хорошим инструментом проектирования.

С их помощью разработчики могут видеть и планировать структуру системы еще до фактического написания кода, благодаря чему они с самого начала могут понять, хорошо ли спроектирована система.

3. ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

3.1. Построение диаграммы классов

1. Откройте Главную диаграмму классов (окно Browser > Logical View >Main) или создайте новую диаграмму (.Logical View >New>Class Diagram). (рис.20)

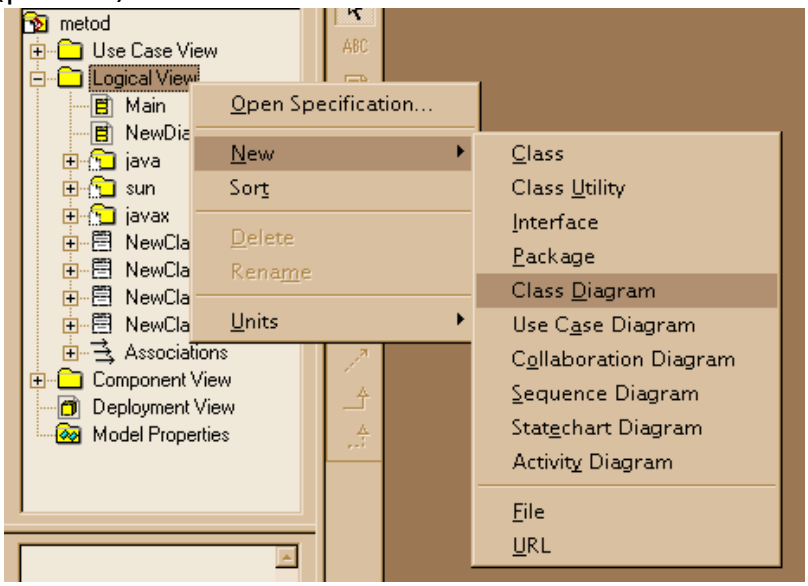

Рис.15

2. Для того чтобы создать пакеты, переносят их непосредственно на рабочий стол Rational Rose из строки инструментов текущей диаграммы, или выполняют последовательность: Logical View>New>Package. Пакеты соединяют стрелками, если необходимо показать их связи.

3. Внутри каждого пакета можно создать вложенную диаграмму классов. Для этого можно щелкнуть мышкой на значке пакета на рабочем столе или в окне броузера выполнить последовательность <Package>>New> Class Diagram.

4. Для того чтобы создать классы, переносят их непосредственно на рабочий стол Rational Rose из строки инструментов текущей диаграммы, выполняют последовательность: Logical View>New>Class (для общих классов), создают класс в конкретном пакете (<Package> >New>Class) или перетаскивают уже существующий класс из окна броузера на рабочий стол.

5. Спецификации, атрибуты и операции классов можно задать из контекстного меню на рабочем столе или в окне броузера (Open

Specifikation, New Attribute и New Operation).

6. Задайте связи между классами.

3.2. Порядок выполнения лабораторной работы

1. Изучите структуру информационной системы.
2. Постройте диаграмму пакетов и диаграммы классов.
3. Ответьте на контрольные вопросы.

4. Контрольные вопросы

1. Что такое диаграмма классов?
2. Назовите основные элементы диаграммы классов.
3. Чем представлена структура класса?
4. Что понимается под атрибутом в диаграмме классов?
5. Что понимается под операцией в диаграмме классов?
6. Какой кванторы видимости могут иметь атрибуты?
7. Какие типы связей поддерживаются в диаграммах классов?
8. Что такое множественность класса и мощность отношений?

ЛАБОРАТОРНАЯ РАБОТА №3

“Динамика поведения информационной системы. Диаграммы взаимодействия"

1. Цель работы

Целью работы является исследование динамики поведения информационной системы и изучение процесса построения диаграмм взаимодействия в заданной предметной области с помощью пакета Rational Rose.

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1. Диаграммы взаимодействия

Диаграммы взаимодействия отображают один из процессов обработки информации в рамках варианта использования. В варианте использования может быть несколько альтернативных потоков. Это значит, что для данного варианта использования нужно создать несколько диаграмм взаимодействия, отражающих один и тот же процесс в различных условиях (одна показывает, что происходит, когда все в порядке, другая, что произойдет в случае ошибки и т.д.).

Диаграммы взаимодействия делятся на диаграммы последовательности (Sequence diagram) и кооперативные диаграммы (Collaboration diagram).

На диаграммах обоих типов может быть представлена одна и та же информация, однако диаграммы последовательности заостряют внимание на управлении, а кооперативные отображают потоки данных.

Диаграммы последовательности упорядочены во времени. Они полезны для того, чтобы понять логическую последовательность событий. Кооперативные диаграммы показывают, как компоненты системы взаимодействуют друг с другом.

Диаграммы взаимодействия содержат:

– Объекты,

– сообщения - с помощью сообщения один объект или класс запрашивает у другого выполнения какой-то конкретной функции, например, форма может запросить у объекта отчет напечатать ее.

Главное здесь: объекты, которые должны быть созданы для реализации функциональных возможностей, заложенных в вариант использования. На диаграммах последовательности и кооперативных диаграммах могут быть показаны объекты, классы или то и другое вместе.

На диаграмме последовательности взаимодействие изображается в виде двухмерной схемы (в формате графа или сети). По вертикали проходит временная ось, где течение времени

происходит сверху вниз. По горизонтали указываются роли классификатора, которые представляют отдельные объекты кооперации. У каждой роли классификатора есть «линия жизни», идущая сверху вниз. Тот период времени, в течение которого объект существует, изображается на диаграмме вертикальной пунктирной линией. Во время вызова процедуры определенного объекта (активизации) его линия жизни изображается двойной линией.

Сообщение выглядит на диаграмме как стрелка, идущая от линии жизни одного объекта к линии жизни другого объекта. Стрелки организованы согласно временной последовательности сообщений.

Активацией называется выполнение процедуры, включающее в себя время ожидания выполнения всех вложенных процедур. На диаграмме активация изображается в виде двойной линии. Которая замещает собой часть линии жизни объекта. Активным называется объект, которому принадлежит стек активаций (и вызовов операций). У каждого активного объекта есть свой собственный поток управления, который выполняется параллельно с другими активными объектами. Объекты, вызываемые пассивными объектами, называется пассивными. Они получают управление только на время вызова, а потом возвращают его.

Диаграмма кооперации – это диаграмма классов, на которой отображаются не просто классификаторы и ассоциации, а роли классификатора и роли в ассоциации. Роли классификатора и роли в ассоциации описывают конфигурацию объектов и связей, которые могут образоваться при выполнении кооперации в реальной системе.

На рис.21 представлена диаграмма последовательности, описывающая процесс поиска прецедентов описаний программных продуктов в базе знаний прецедентов.

Работая с базой знаний прецедентов, пользователь вводит простой запрос, содержащий набор признаков, описывающих объект автоматизации – спецификации заказа на программный продукт. Сначала признаки заказа сравниваются с признаками концептов – классов прецедентов, выбирается наиболее подходящий. Затем поиск ведется уже в классе прецедентов. Если система находит прецедент (несколько прецедентов), выводится решение об использовании данного прецедента и правило его адаптации к требованиям заказчика. Если система не находит ни одного прецедента, новый прецедент записывается в базу знаний.

На рис.22 представлена диаграмма кооперации, описывающая тот же процесс.

С помощью диаграмм Взаимодействия проектировщики и разработчики системы могут определить классы, которые нужно создать, связи между ними, а также операции и ответственности (responsibilities) каждого класса. Диаграммы Взаимодействия — краеугольный камень, на котором возводится оставшаяся часть проекта.

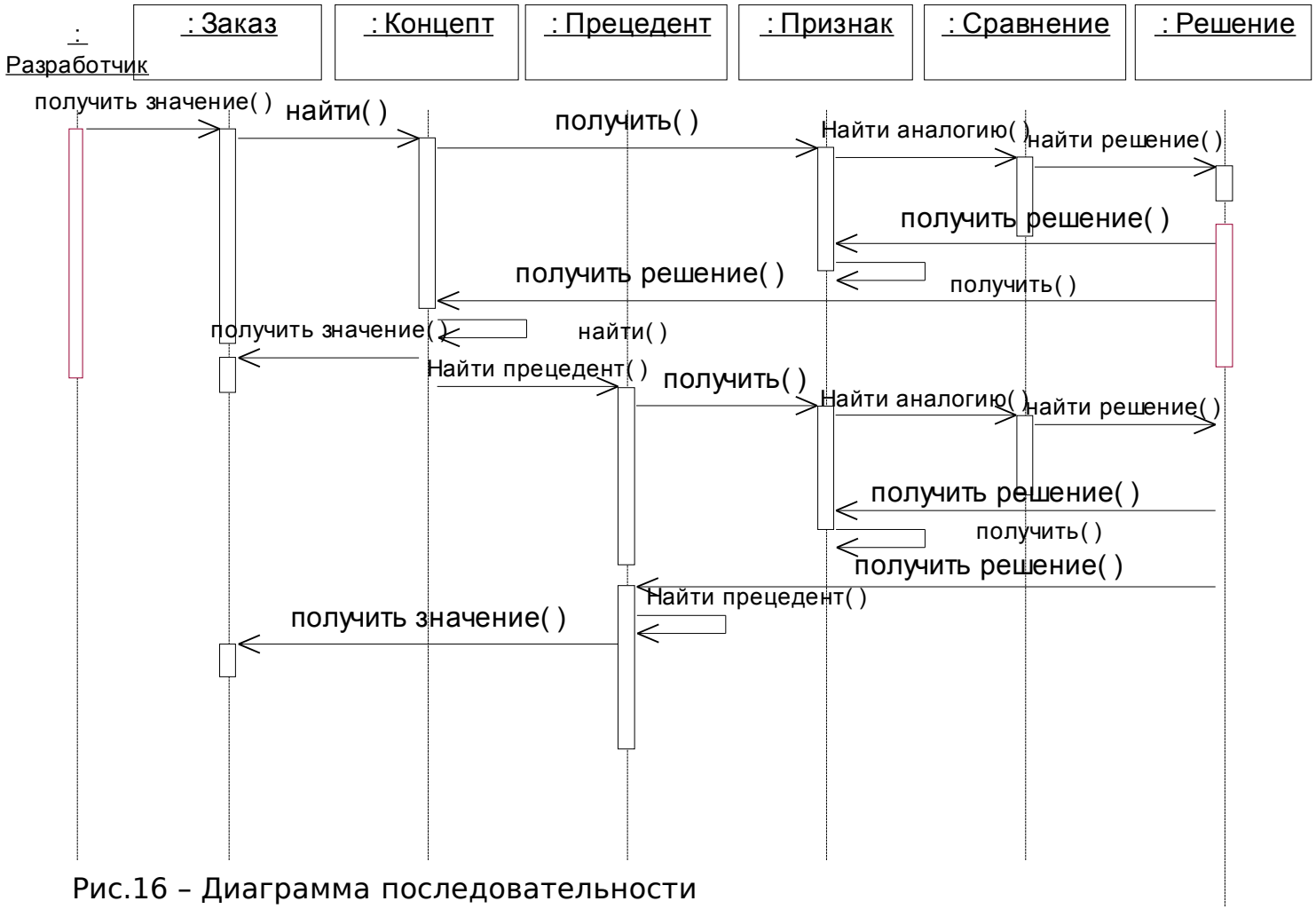

Рис.16 – Диаграмма последовательности

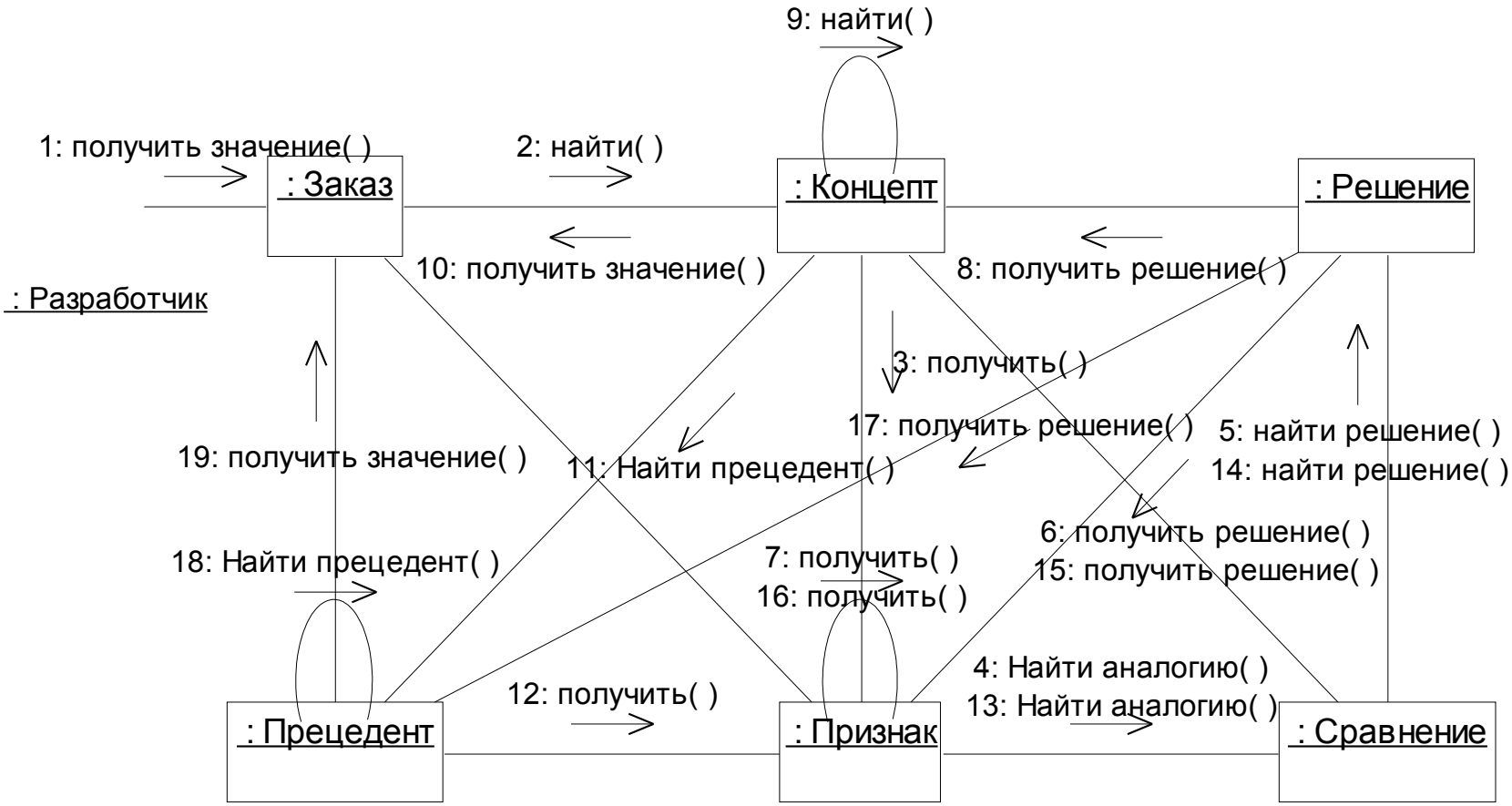

Рис.17 – Диаграмма кооперации

3. ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

3.1. Построение диаграмм взаимодействий

1. Создайте диаграмму последовательности для выбранного прецедента (контекстное меню в броузере> New > Sequence Diagram или в контекстном меню на рабочем столе > Sub Diagram> New Sequence Diagram).
2. Создайте объекты, поместив их непосредственно на рабочий стол из строки инструментов (Object), или перенесите из окна броузера уже существующий объект.
3. Добавьте необходимые сообщения (Message) из строки инструментов. Вызовите окно спецификаций сообщения, щелкнув по линии мышкой. Задайте имя и свойства.
4. Диаграмму кооперацийможно создать из диаграммы последовательности, если она отображает тот же процесс (Browse > Create Collaboration Diagram).

3.2. Порядок выполнения лабораторной работы

1. Исследуйте динамику какого-либо варианта использования.
2. Постройте диаграмму последовательности.
3. Из диаграммы последовательности получите диаграмму кооперации.
4. Ответьте на контрольные вопросы.

4. Контрольные вопросы

5. Для чего строятся диаграммы взаимодействия?
6. Как изображается взаимодействие на диаграмме последовательности?
7. Что такое активация?
8. В чем отличие диаграммы последовательностей и диаграммы кооперацийй?
9. Перечислите основные элементы диаграмм взаимодействий.

ЛАБОРАТОРНАЯ РАБОТА №4

“Динамика поведения информационной системы. Диаграммы состояний. Физическая модель системы. Диаграммы реализации"

1. Цель работы

Целью работы является исследование динамики поведения информационной системы и физической модели системы, изучение процесса построения диаграммы состояний и диаграмм реализации в заданной предметной области с помощью пакета Rational Rose.

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1. Диаграммы состояний

Диаграмма состояний (statechart diagram) показывает положение одиночного объекта, события или сообщения, которые вызывают переходы из одного состояния в другое, и действия, являющиеся результатом смены состояний.

На диаграмме состояний отображают жизненный цикл одного объекта, начиная с момента его создания и заканчивая разрушением. С помощью таких диаграмм удобно моделировать динамику поведения класса. Как правило, диаграммы состояний не требуется создавать для каждого класса. Диаграмма состояний создается для классов с динамическим поведением. Многие проекты вообще обходятся без них.

Состояние – это некоторое положение в жизни объекта, при котором он удовлетворяет определенному условию, выполняет некоторое действие или ожидает события.

Диаграмма состояний включает все сообщения, которые объект получает и отправляет. Сценарий – это одиночный проход по диаграмме состояний. Интервал между двумя сообщениями, отправляемыми объектом обычно представляет состояние. Конечный автомат представляет собой глубокий взгляд на поведение конкретного объекта. Спецификация автомата настолько подробна, что годится для непосредственной реализации в программном коде. Однако конечные автоматы описывают только единичные объекты, а не их совокупность.

Переходы между состояниями представляют собой смену исходного состояния последующим (которое может быть тем же, что и исходное). Переход может сопровождаться определенным действием.

С переходом между состояниями может быть связано условие (guard condition) и/или определенное действие (action). Переход может также вызывать событие (event). Действие – это поведение, проявляющееся при возникновении перехода. Событие – это сообщение, отправляемое другому объекту системы. Условие – булево

выражение значений атрибутов, которое допускает переход, только если оно верно. И действие, и проверка условия представляют собой поведение объекта и обычно реализуются в виде операций.

Действия, сопровождающие переходы в определенное состояние, можно рассматривать как входные действия (entry action) для этого состояния. И наоборот, действия, сопровождающие переходы из данного состояния, являются для него выходными (exit action). Поведение, возникающее внутри состояния, называется деятельностью (activity). Деятельность начинается при входе в состояние и завершается или прерывается при выходе из него.

Переходом (transition) называется перемещение из одного состояния в другое. Совокупность переходов диаграммы показывает, как объект может переходить из одного состояния в другое. На диаграмме переходы изображают в виде стрелки, начинающейся в начальном и заканчивающейся в последующем состоянии. Переходы могут быть рефлексивными: объект переходит в то же состояние, в котором он в настоящий момент находится. Рефлексивные переходы изображают в виде стрелки, начинающейся и завершающейся на одном и том же состоянии.

Для уменьшения беспорядка на диаграмме можно вкладывать состояния одно в другое. Вложенные состояния называются подсостояниями (substates), а те, в которые они вложены, — суперсостояниями (superstates).

Если у нескольких состояний имеются идентичные переходы, эти состояния можно сгруппировать вместе в суперсостояние. Затем, вместо того чтобы поддерживать одинаковые переходы (по одному на каждое состояние), их можно объединить, перенеся в суперсостояние.

Суперсостояния позволяют навести порядок на диаграмме Состояний.

Для анализа динамики поведения класса необходимо рассмотреть его атрибуты. Экземпляр класса может вести себя по-разному в зависимости от их значений.

Полезно исследовать связи между классами. Рассмотрите все связи, множественность которых может принимать нулевое значение. Нули указывают, что данная связь не является обязательной. Ведет ли себя экземпляр класса по-разному при наличии и отсутствии связи? Если да, то он имеет несколько состояний.

В среде Rose на основании диаграмм Состояний не генерируется никакого исходного кода. Они нужны для того, чтобы документировать динамику поведения класса, благодаря чему аналитики и разработчики получат о нем четкое представление. Реализацией заложенной в эти диаграммы логики будут заниматься разработчики. Как и в случае других диаграмм UML, диаграммы Состояний дают возможность обсудить и документировать логику перед началом процесса кодирования.

2.2. Диаграммы компонентов

Компонент – физический модуль кода. Это могут быть библиотеки исходного кода и исполняемые файлы. Компонента – исходный код, бинарный код или run-time объект.

Диаграммы компонентов показывают, как выглядит модель на физическом уровне. На ней изображаются компоненты программного обеспечения системы и связи между ними.

Диаграммы компонентов отображают типы компонент и зависимости между программными компонентами, возникающие на этапе компиляции или в процессе выполнения программы, в частности связь файлов исходного кода с динамическими библиотеками DLL. На диаграммах компонентов изображается вхождение классов и объектов в программные компоненты системы (модули, библиотеки и т.д.).

Главная диаграмма компонентов обычно представляет определенные для системы пакеты.

Рассмотрим пример. Базу прецедентов в системе поддержки принятия решений при адаптации и внедрении программных продуктов предложено реализовать с помощью системы управления базами данных Microsoft Access, поэтому на диаграмме показано взаимодействие компонента «База прецедентов» с компонентами «Библиотеки Microsoft Access» и «Среда Microsoft Access».

Интерфейс пользователя было решено реализовать в виде Web-страниц.

Диаграмма компонентов для описания базы прецедентов представлена на рис. 20.

Компоненты исходного кода – это программные файлы, содержащиеся внутри пакетов. Классы в логическом представлении отображаются на компоненты в представлении компонентов.

Представление процессов отображает структуру программной реализации системы. Представление процессов учитывает такие потребности. Как производительность, надежность, масштабируемость, целостность, управление системой и синхронизация. Компоненты связаны отношением зависимости. Программные компоненты отображают классы на программные библиотеки. Такие как Java – applet, Active-X и динамические библиотеки.

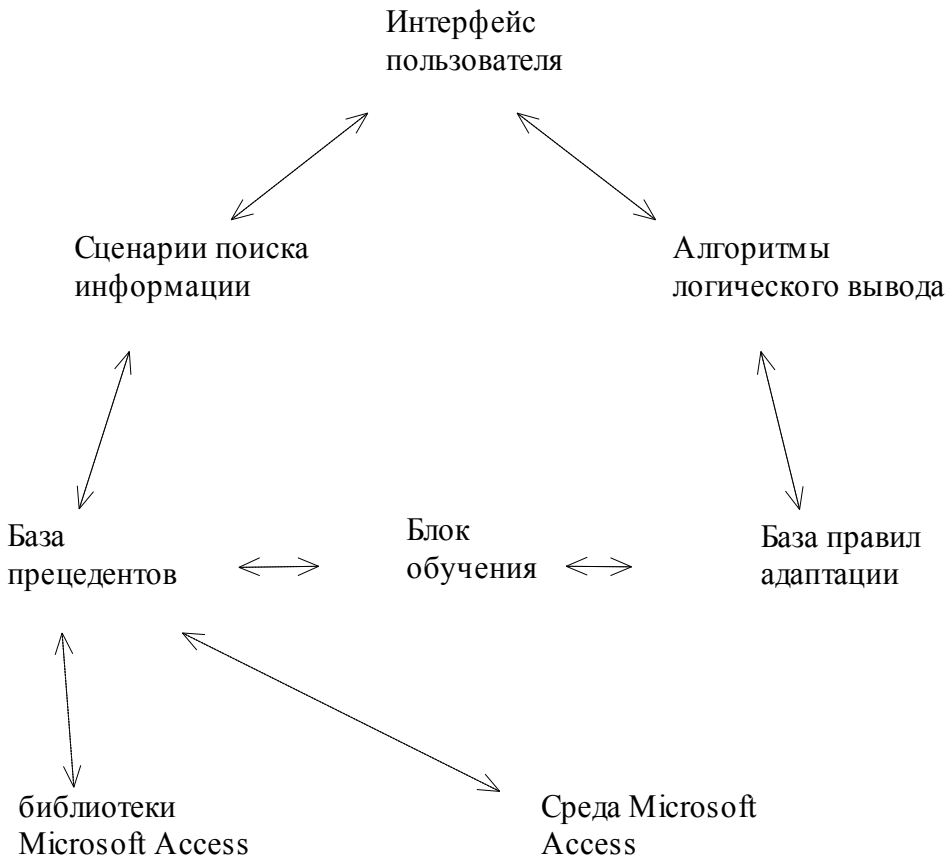

Рис.20 –Диаграмма компонентов

2.3. Диаграммы размещения

Диаграммы размещения показывают физическое размещение различных компонент системы в сети. При помощи диаграмм развертывания документируется размещение программных модулей на узлах (физических и логических устройствах) системы.

Процессор – любая машина, имеющая вычислительную мощность, т.е. способность производить обработку данных. В эту категорию попадают серверы, рабочие станции и другие устройства, содержащие физические процессоры.

Устройство – аппаратура, не обладающая вычислительной мощностью. Это, например терминалы ввода/вывода, принтеры, сканеры.

На рис.21 представлена диаграмма размещений системы поддержки принятия решений при адаптации и внедрении программных продуктов. Было предложено установить базу прецедентов и базу правил адаптации на сервере знаний. По сети

Intranet ПК эксперта и разработчика посредством броузера соединяются с сервером. Используется HTTP-интерфейс.

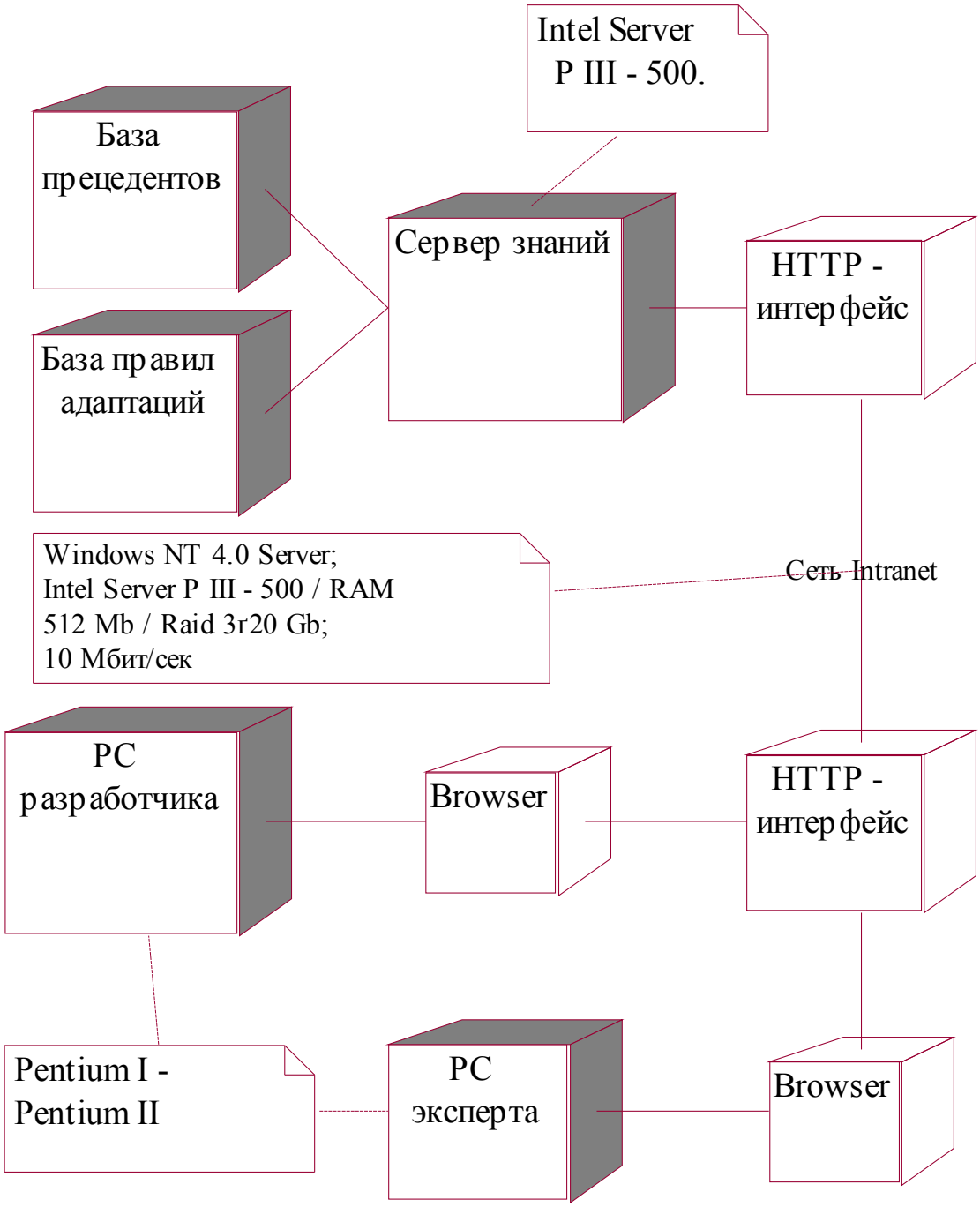

Рис.21 –Диаграмма размещения

Эксперт контролирует базы знаний, следит за достоверностью информации, добавляет новые правила адаптации, редактирует

данные о программных продуктах, удаляет «устаревшие» или неиспользуемые данные.

Разработчик вводит признаки новой системы, получает информацию о выбранном СППР программном продукте или список наиболее подходящих функциональных модулей, предложения по адаптации, прогноз стоимости, времени на настройку и внедрение.

2.4. Тестирование проекта

3. ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

1.1. Построение диаграммы состояний

1. Создайте новую диаграмму (.Component View >New>Component Diagram).
2. Создайте компоненты, поместив их непосредственно на рабочий стол из строки инструментов.
3. Добавьте необходимые связи между компонентами.

1.2. Построение диаграммы компонентов

1. Откройте Главную диаграмму компонентов (окно Browser > Component View >Main) или создайте новую диаграмму (.Component View >New>Component Diagram).
2. Создайте компоненты, поместив их непосредственно на рабочий стол из строки инструментов.
3. Добавьте необходимые связи между компонентами.

3.3. Порядок выполнения лабораторной работы

1. Постройте диаграмму состояний.
2. Постройте диаграмму компонентов.
3. Постройте диаграмму размещения.
4. Протестируйте проект.
5. Ответьте на контрольные вопросы.

4. Контрольные вопросы

1. Для чего строятся диаграммы состояний?
2. Назовите основные компоненты диаграммы состояний.
3. Что такое входные и выходные действия.
4. Для чего строятся диаграммы компонентов?
5. Что такое диаграмма размещений.

Литература

1. Буч Г., Рамбо Д, Джекобсон А. Язык UML. Руководство пользователя: Пер.с.англ.-М.:ДМК,200. – 432 с.
2. Боггс У, Боггс М. UML и Rational Rose. Пер с англ. – М.: Издательство «Лори», 2000. – 580 с.
3. Кватрани Т. Rational Rose 2000 и UML. Визуальное моделирование: Пер. с англ. – М.: ДМК Пресс, 2001. – 176 с.: ил. (Серия «Объектно-ориентированные технологии в программировании»).
4. Крачтен Филипп. Введение в Rational Unified Process. 2-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2002. – 240 с.: ил. – Парал. тит. англ.
5. Ларман Крэг. Применение UML и шаблонов проектирования. : Пер. с англ. : Уч. пос. – М.: Издательский дом «Вильямс», 2001. – 496 с.: ил. – Парал. тит. англ.
6. Леоненков А.В. Самоучитель UML. – СПб.: БХВ-Петербург, 2001. – 304 с.: ил.
7. Трофимов С.А. Case-технологии: практическая работа в Rational Rose. – М.: ЗАО «Издательство БИНОМ», 2001. – 272 с.: ил.
8. Фаулер М., Скотт К. UML в кратком изложении. Применение стандартного языка объектного моделирования: Пер. с англ. – М.: Мир, 1999. – 191 с.: ил.