

Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

О. В. Бугай, В. С. Юденков

САПР ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИЗДАТЕЛЬСКО-ПОЛИГРАФИЧЕСКОГО КОМПЛЕКСА

*Допущено
Министерством образования Республики Беларусь
в качестве учебного пособия для сту-
дентов высших учебных
заведений по специальностям «Программное обеспечение информационных
технологий, «Информационные системы и технологии
(издательско-полиграфический комплекс)»*

Минск 2008

УДК 004.434:655(075.8)

ББК 32.973я7

Б90

Рецензенты:

кафедра математических проблем управления учреждения образования «Гомельский государственный университет имени Франциска Скорины» (заведующий кафедрой профессор, доктор технических наук *И. В. Максимей*);
заведующий кафедрой систем автоматизированного проектирования Белорусского национального технического университета доцент, кандидат технических наук *В. А. Кочуров*

Все права на данное издание защищены. Воспроизведение всей книги или ее части не может быть осуществлено без разрешения учреждения образования «Белорусский государственный технологический университет».

Бугай, О. В.

Б90 САПР программного обеспечения издательско-полиграфического комплекса : учеб. пособие для студентов специальностей «Программное обеспечение информационных технологий», «Информационные системы и технологии (издательско-полиграфический комплекс)» / О. В. Бугай, В. С. Юденков. – Минск : БГТУ, 2008. – 174 с.

ISBN 978-985-434-784-4.

В систематизированном виде приведены сведения по DF-, IDEF0-, IDEF3-, IDEF1X-, UML-диаграммам и особенностям их построения с использованием современных CASE-средств. Показан пример разработки конкретного приложения из области полиграфии. Рассмотрен графический редактор VISIO как альтернатива CASE-пакетам.

Учебное пособие может быть полезно студентам и аспирантам, которые занимаются проблемами программного обеспечения для моделирования и его применения при проектировании информационных систем.

УДК 004.434:655(075.8)

ББК 32.973я7

ISBN 978-985-434-784-4

© УО «Белорусский государственный технологический университет», 2008

© Бугай О. В., Юденков В. С., 2008

ПРЕДИСЛОВИЕ

С увеличением сложности программного обеспечения (ПО) возникла крайняя необходимость процесс его разработки начинать с построения модели программы, по которой в дальнейшем осуществляется ее реализация.

Для моделирования ПО прибегают к двум диаграммным техникам: структурного системного анализа и проектирования, объектно-ориентированного анализа и проектирования. В пособии дается краткое описание основных диаграммных техник как одного, так и другого подхода и их инструментальной поддержки. Приводится сквозной пример их применения на всех стадиях разработки конкретного ПО из области полиграфии: моделирование бизнес-процессов, данных, объектно-ориентированного анализа и проектирования, а также создание каркаса программы. Материал пособия самодостаточен для подготовки читателя к самостоятельному решению задач, связанных с моделированием разрабатываемых программ путем использования современных методологий и средств автоматизации разработки ПО.

Учебное пособие состоит из 18 разделов, каждый из которых посвящен определенной теме. Все они постепенно ведут читателя по этапам разработки приложения с использованием современных техник моделирования и инструментария его поддержки.

В разделе «САПР в контексте разработки программного обеспечения» даются общие сведения о жизненном цикле ПО, роли САПР в его цикле и инструментальной поддержке САПР ПО.

Начиная с раздела «Моделирование бизнес-процессов» приводятся краткие сведения по рекомендуемой диаграммной технике, особенностям ее построения с использованием рекомендуемого CASE-средства, например моделирования, а также рассматривается применение возможностей графического редактора VISIO как альтернативы.

Раздел «Моделирование бизнес-процессов» посвящен высокоуровневому анализу бизнес-процессов с построением моделей AS – IS и TO – BE на примере, связанном с полиграфией, с применением CASE-средства AllFusion Process Modeler. Указывается на возможность использования редактора VISIO как альтернативы.

В разделе «Моделирование данных» акцентируется внимание на моделировании данных для вышеприведенного примера с применением CASE-средства AllFusion ERwin Data Modeler.

В разделах «Объектно-ориентированный подход к разработке программного обеспечения», «Модель вариантов использования и ее разработка», «Дальнейшее развитие модели программного обеспечения», «Кодогенерация» соответственно рассматривается объектно-ориентированный анализ, проектирование и кодогенерация приложения для упомянутого примера с применением CASE-средства Rational Rose. Указывается на возможность использования редактора VISIO как альтернативы.

В заключение авторы хотят выразить благодарность Надежде Жиляк, принявшей активное участие в разработке программ по теме «Информационная система обработки полиграфических заказов».

УСЛОВНЫЕ ОБОЗНАЧЕНИЯ

Для сокращения описания особенностей использования применяемых CASE-средств принят ряд условных обозначений.

Имя исходного пункта – Окно Browser:, Menu:, Панель Diagram и др.;

RClick – нажать правую кнопку мыши;

DbClick – двойной щелчок мышью;

→ – выбрать;

= – присвоить значение;

← = – присвоить принятому значению по умолчанию новое значение;

| – ИЛИ;

! – И;

[Имя кнопки] – нажать кнопку с заданным именем;

LClick_∨ – нажать левую кнопку мыши;

LClick_^ – отпустить левую кнопку мыши;

{ вар 1 | вар 2 | ... } – альтернатива;

1 { }* – повторить от 1-го до необходимого количества раз;

() – необязательный элемент (параметр).

1. САПР В КОНТЕКСТЕ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1.1. Жизненный цикл программного обеспечения и его критические этапы

В основе деятельности по созданию и использованию ПО лежит понятие его жизненного цикла (ЖЦ).

Традиционно выделяют следующие этапы жизненного цикла ПО:

- анализ требований;
- проектирование;
- реализация;
- тестирование и отладка;
- эксплуатация и сопровождение.

Жизненный цикл образуется в соответствии с принципом нисходящего проектирования и, как правило, носит итерационный характер. Порядок исполнения этапов, а также критерии перехода от этапа к этапу определяют модели жизненного цикла.

Наибольшее распространение получили три следующие модели жизненного цикла.

1. *Каскадная модель (70–80-е гг.)* характеризуется переходом на следующий этап после полного завершения работ по предыдущему этапу (рис. 1.1).

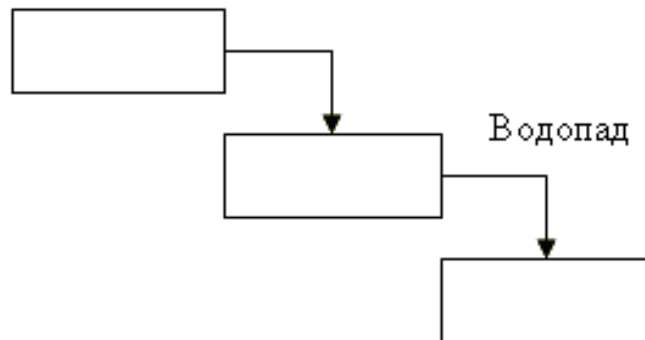


Рис. 1.1. Каскадная модель

2. *Каскадно-итерационная модель (1980–1985 гг.)* с циклом обратной связи между всеми этапами (рис. 1.2).

Это обеспечивает снижение трудозатрат по сравнению с каскадной моделью. Однако время жизни каждого из этапов растягивается на весь период разработки.

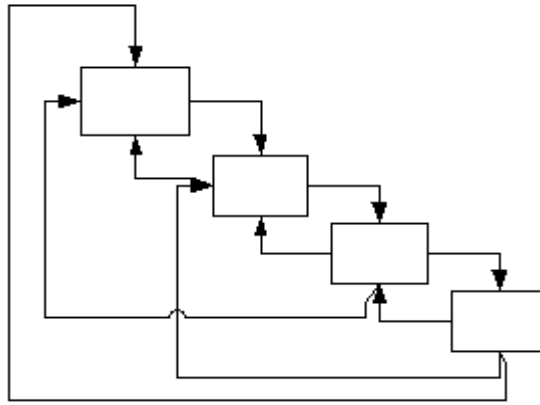


Рис. 1.2. Каскадно-итерационная модель

3. *Спиральная модель (начиная с 1986 г.) с упором на начальные этапы жизненного цикла (анализ и проектирование) (рис. 1.3).*

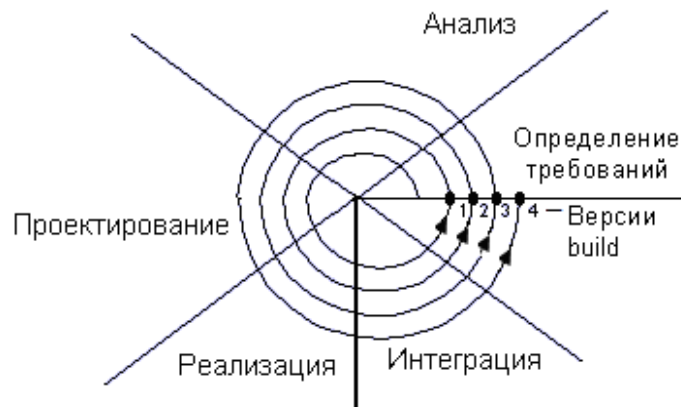


Рис. 1.3. Спиральная модель

При этом с переходом на очередной виток углубляются и последовательно корректируются детали проекта.

Анализ и проектирование являются наиболее важными этапами и позволяют построить модель ПО, используемую для его программирования, уточняются, формализуются и документируются. Этот этап отвечает на вопрос: «Что должна делать будущая система?».

Этап проектирования дает ответ на вопрос: «Как система должна удовлетворять предъявляемым требованиям?».

В результате выполнения этапов анализа и проектирования должен быть получен проект (модель системы либо только ее части), содержащий достаточно информации для реализации системы или же части, в рамках бюджета выделенных ресурсов и времени.

1.2. САПР программного обеспечения в жизненном цикле программного обеспечения

Технология создания информационных систем (ИС) предъявляет особые требования к методам реализации и инструментальным средствам [3, 4].

Во-первых, как было сказано выше, процесс разработки ИС при структурном подходе включает в себя стадию анализа (описание бизнес-логики предметной области с построением модели AS – IS и определение способа ее оптимизации с созданием модели TO – BE), проектирования (определение структуры и архитектуры будущей системы), кодирования, тестирования и сопровождения. При объектно-ориентированном же подходе создание моделей AS – IS и TO – BE так же актуально, как и в первом случае, поскольку позволяет перейти к грамотному описанию технического задания на предмет разработки в виде модели вариантов использования, затем к планированию итерации конструирования, каждая из которых включает логическое и физическое моделирование, реализацию и тестирование, а после завершения процесса конструирования – сопровождение. Известно, что исправление ошибок, допущенных на предыдущей стадии, обходится в 10 раз дороже, чем на текущей, поэтому наиболее критическими являются первые стадии проекта. По этой причине весьма важно иметь эффективные средства автоматизации ранних этапов реализации проекта.

Во-вторых, проект по созданию сложной ИС требует коллективной разработки. Коллективная работа над крупными проектами нуждается в средствах координации и управления коллективом разработчиков. Это может быть достигнуто за счет организации и поддержки этими средствами репозитория, обеспечивающего хранение, доступ, обновление, анализ и визуализацию всей информации по проекту ИС. Содержимое репозитория включает информационные объекты различных типов, отношения между компонентами, правила использования или обработки этих компонентов. Каждый информационный объект в репозитории описывается перечислением его свойств: идентификатор, имена-синонимы, тип, текстовое описание, компоненты, файл-хранилище, область значений и т. д. На основе репозитория реализуется интеграция CASE-средств и разделение информации между разработчиками. При этом обеспечивается несколько уровней интеграции:

- общий пользовательский интерфейс по всем средствам;
- передача данных между средствами;
- интеграция этапов разработки через единую систему представлений стадий ЖЦ;
- передача данных и средств между аппаратными платформами.

Репозиторий является базой для стандартизации документов по проекту и для контроля состоятельности проектных спецификаций. Все отчеты по содержанию, перекрестным ссылкам, результатам анализа и декомпозиции строятся автоматически по репозиторию. Важные функции управления и контроля также реализуются на основе репозитария. Из сказанного следует, что основой средств координации и управления коллективом разработчиков должны быть репозитарии или база данных разработки (БДР).

В-третьих, прошли те времена, когда программа «переживала» несколько поколений программистов. Ныне жизненный цикл создания сложной ИС сопоставим с ожидаемым временем ее эксплуатации; в современных условиях компании перестраивают свои бизнес-процессы примерно раз в два года. Столько же требуется времени, если следовать традиционной технологии для создания ИС. При таком подходе может оказаться, что к моменту сдачи готовой ИС она уже никому не нужна, поскольку компания-заказчик вынуждена перейти на новую технологию производственной деятельности. Следовательно, для создания ИС крайне необходим инструмент, существенно (в несколько раз) уменьшающий время разработки ИС.

В-четвертых, при продолжительном жизненном цикле может оказаться, что в процессе создания системы внешние условия изменились. Как правило, внешние изменения в проекте на поздних этапах разработки ИС (даже если разработчики постоянно задают рекомендуемый вопрос: «А что если потребуются...?») – крайне трудоемкий и дорогостоящий процесс.

Видим, что в современных условиях к САПР ПО предъявляются весьма высокие требования, часто весьма несовместимые между собой. Что касается факта их востребованности, то он очевиден.

1.3. Инструментальная поддержка автоматизации разработки программного обеспечения

Автоматизация разработки ПО с помощью различных инструментальных средств позволяет ускорить и упростить разработку

ПО, а также обеспечить достижение качественных показателей. Многочисленные инструментальные средства можно разделить на три группы:

- 1) общецелевые средства автоматизации программирования;
- 2) инструментальные средства, используемые на определенном этапе разработки ПО либо для создания тех или иных компонентов;
- 3) специализированные инструментальные системы.

К общецелевым средствам автоматизации программирования относятся языки программирования высокого уровня, файловые системы, библиотеки стандартных программ, редакторы текстов, загрузчики, системы отладки программ и другие компоненты ПО. Эти универсальные инструментальные средства могут применяться для создания самого разнообразного ПО. Они существенно облегчают процесс разработки ПО и обеспечивают поддержку ряда эффективных технологий проектирования, программирования, тестирования и эксплуатации ПО. Однако общецелевые средства решают проблему автоматизации лишь частично, так как, будучи универсальными, не учитывают структурные и функциональные особенности программных изделий (ПИ), и эта задача ложится на разработчика.

Отдельные инструментальные средства используются для поддержки того или иного этапа создания и эксплуатации ПО. Эти средства не входят в штатное ПО ЭВМ и реализованы в виде программных комплексов. К ним относятся:

- интегрированные среды разработки (ИСР);
- документаторы;
- системы автоматизированных преобразований;
- системы генерации компиляторов и многие другие.

Значительная универсальность и разнообразие средств этой группы позволяют исполнителям включать в технологические линии по разработке ПО средства, наилучшим образом соответствующие особенностям создаваемого в конкретном случае программного изделия (ПИ).

Специальные инструментальные системы или так называемые технологические комплексы дают возможность автоматизировать выполнение различных видов деятельности на всех стадиях жизненного цикла ПО или некоторой их части (ЖЦ ПО). В развитых технологических комплексах автоматизированы:

- 1) создание проектной информации и занесение ее в базу данных разработки (БДР) на всех стадиях разработки ПИ;

- 2) автономное тестирование подпрограмм;
- 3) анализ правильности, согласованности и замкнутости проектной информации определенной стадии;
- 4) оценка степени завершенности разработки и качества выполнения работ;
- 5) планирование хода разработки и контроль за выполнением планов;
- 6) общая организация работ в соответствии с некоторой технологической схемой;
- 7) оформление технологической и эксплуатационной документации.

Использование технологических комплексов имеет издержки, так как требует больших временных затрат на освоение и достаточно много ресурсов инструментальных ЭВМ. Поэтому технологические комплексы эффективны, начиная с определенного уровня сложности и суммарного объема создаваемого ПО. При разработке программ малого и среднего размера предпочтительно применять технологические комплексы с небольшим набором автоматизируемых видов деятельности.

Технологические комплексы отличаются универсальностью. Каждый из них ориентирован на разработку того или иного класса ПО по определенным родственным технологиям.

Технология создания ПО, в том числе и экспертных систем, предъявляет особые требования к методам реализации и программным инструментальным средствам:

- эффективная автоматизация ранних этапов реализации проекта;
- координация и управление коллективом разработчиков;
- существенное уменьшение времени разработки;
- гибкость к изменяющимся условиям.

Этим требованиям наиболее полно соответствуют средства, используемые при применении таких высокоэффективных технологий, как CASE-технологии. CASE (Computer-Aided Software/System Engineering) – это технология, представляющая собой совокупность методологий, анализа, проектирования, разработки и сопровождения сложных систем ПО, поддержанная комплексом взаимосвязанных средств автоматизации. CASE – это инструментарий для системных аналитиков, разработчиков и программистов, позволяющий автоматизировать процесс проектирования и разработки ПО. Основными покупателями CASE-пакетов за рубежом

являются военные организации, центры обработки данных и коммерческие фирмы по разработке ПО.

CASE – не революция в программотехнике, а результат эволюционного развития технологических средств разработки ПО.

CASE-технологии не являются самостоятельными методологиями. Они только развивают структурные и объектно-ориентированные методологии, делают более эффективными их применение за счет автоматизации.

Практически ни один серьезный проект не осуществляется без использования CASE-средств, основные методы которых:

- 1) лучшие из известных методов структурирования системного анализа и проектирования;
- 2) объектно-ориентированный подход;
- 3) повторное использование компонентов (ПИК);
- 4) реинженерия (перепрограммирование компонентов);
- 5) инженерия качества (ГОСТы);
- 6) автоматическая генерация выходного кода.

Ниже в таблице дана сравнительная оценка трудозатрат по фазам ЖЦ.

Таблица

Сравнительная оценка трудозатрат по фазам жизненного цикла, %

| Технология создания ПО | Анализ | Проектирование | Кодирование | Тестирование |
|-------------------------|--------|----------------|-------------|--------------|
| Традиционная разработка | 20 | 15 | 20 | 45 |
| Структурная методология | 30 | 30 | 15 | 25 |
| CASE-технологии | 40 | 40 | 5 | 15 |

1.4. Концептуальные основы CASE-технологий

Эволюция CASE-средств. Традиционно выделяют шесть периодов, качественно отличающихся применяемой техникой и методами разработки ПО. Они характеризуются использованием следующих инструментальных средств [1, 2]:

- ассемблеров, дампов памяти, анализаторов;
- компиляторов, интерпретаторов, трассировщиков;
- символических отладчиков пакетов программ;
- систем анализа и управления исходными текстами;

– CASE-средств анализа требований, проектирования спецификаций и структуры; редактирования интерфейсов (CASE-I);

– CASE-средств генерации исходных текстов и реализации интегрированного окружения полного ЖЦ ПО (CASE-II).

CASE-I поддерживает ЖЦ частично и концентрирует внимание на системном анализе, определении требований, системном проектировании и логическом проектировании БД.

CASE-II отличается значительно более развитыми возможностями, улучшенными характеристиками и исчерпывающим подходом к полному ЖЦ. Может включать свыше 100 функциональных компонентов, поддерживающих все стадии ЖЦ.

CASE-модель ЖЦ ПО. CASE-технологии предлагают новый, основанный на автоматизации подход концепции ЖЦ ПО. Так, если традиционно простейшая модель ЖЦ включает такие стадии, как анализ, проектирование, кодирование, сопровождение, то соответствующая CASE-модель – прототипирование, проектирование спецификаций, контроль проекта, кодогенерацию, системное тестирование, сопровождение.

Наиболее автоматизированными при этом являются стадии контроля проекта и кодогенерации.

1.5. Состав, структура и функциональные особенности CASE-средств

CASE-средства представляют собой новый тип графически ориентированных инструментов, восходящих к системе поддержки ЖЦ ПО.

К ним относится всякое программное средство, обеспечивающее автоматическую помощь при разработке ПО, его сопровождение или управление проектом и проявляющее следующие дополнительные черты:

1) мощная графика для описания и документирования ПО, а также для улучшения интерфейса с разработчиком;

2) интеграция, обеспечивающая легкость передачи данных между средствами и позволяющая управлять всем процессом проектирования и разработки ПО непосредственно через процесс планирования проекта;

3) использование компьютерного хранилища (репозитария) для всей информации о проекте.

Кроме того, в основе концептуального построения CASE-средств лежат следующие положения:

- человеческий фактор, определяющий разработку ПО как легкий, удобный и экономический процесс;
- широкое использование таких базовых программных средств, как БД, СУБД, компиляторы, отладчики, документаторы и тому подобное;
- автоматизированная или автоматическая кодогенерация;
- облегчение сложности компонент;
- достижимость для различных категорий пользователей;
- рентабельность;
- сопровождаемость, обеспечивающая способность адаптации при изменении целей проекта.

Интегрированный CASE-пакет содержит четыре основных компонента:

1) средство централизованного хранения всей информации о проектируемом ПО в течение всего ЖЦ ПО (репозитарий);

2) средство ввода данных в репозитарий, а также для организации взаимодействия с CASE-пакетом;

3) средства анализа, планирования и разработки для обеспечения планирования и анализа различных описаний, а также их преобразования в процессе разработки;

4) средства вывода для документирования управления проектом и кодогенерацией.

Эти компоненты в совокупности должны:

- поддерживать графические модели;
- контролировать ошибки;
- организовывать и поддерживать репозитарий;
- поддерживать процесс проектирования и разработки.

1.6. Классификация CASE-средств

CASE-средства делятся на типы, категории, уровни [1, 2].

Типы CASE-средств. Классификация по типам отражает функциональную ориентацию CASE-средств в технологическом процессе.

Различают шесть основных типов CASE-средств.

1. Анализ и проектирование. Используются для создания спецификаций системы и ее проектирования, ориентированного на функции, данные либо на то и другое.

Их целью является определение системных требований и свойств, которыми должна обладать система, а также создание проекта

системы, удовлетворяющей этим требованиям и обладающей соответствующими свойствами.

К этому типу средств относятся: CASE-аналитик (The Developer), Rational Rose, Prokit *Workbench, BPWin и др.

2. *Проектирование БД и файлов.* Средства данной группы обеспечивают логическое моделирование данных, автоматическое преобразование модели данных в ЗНФ, генерацию схем БД и описаний форматов файлов на уровне программного кода.

К этому типу средств относятся: ErWin, Chen ToolKit, S-Designer, Designer 2000 и др.

3. *Программирование.* Средства этой группы поддаются стадии программирования и тестирования, а также автоматической кодогенерации из спецификаций, получая полностью документируемую выполняемую программу. Помимо диаграммеров, в эту группу отнесены традиционные генераторы кодов, анализаторы кодов, генераторы наборов тестов, анализаторы покрытия тестами, отладчики: COBOL, 2/WorkBench, Decase, Netron/CAP, APS.

4. *Сопровождение и реинжиниринг.* К этим типам средств относятся документаторы, анализаторы программ, средства реконструирования и реинжиниринга: Adpac CASE Tools, Scan/COBOL, INSPECTOR/RECODER.

Их целью является корректировка, изменение, анализ, преобразование и реинжиниринг существующей системы.

Средства этой группы включают:

- статические анализаторы для продуцирования схем системы ПО из ее кодов, оценки влияния модификаций;

- динамические анализаторы (компиляторы и интерпретаторы со встроенными отладочными возможностями);

- документаторы, позволяющие автоматически получить обновленную документацию при изменении кода;

- редакторы кодов, автоматически изменяющие при редактировании все предшествующие коду структуры;

- средства доступа к спецификациям, их модификации и генерация нового кода;

- средства реверсного инжиниринга, транслирующие коды в спецификации.

5. *Окружение.* Это средства поддержки платформ для интеграции, создания и придания товарного вида CASE-средствам. К ним относятся: MultiCam, Design OA.

6. *Управление проектом.* Это средства, поддерживающие планирование, контроль, руководство, взаимодействие, т. е. функции, необходимые в процессе разработки и сопровождения проектов. К ним относятся: Project WorkBench.

Категории CASE-средств. Классификация по категориям определяет уровень интегрированности по выполняемым функциям и включает вспомогательные программы (Tools), пакеты разработчика (ToolKit) и инструментальные средства (WorkBench).

1. *Категория Tools.* Обозначает вспомогательный пакет, который решает небольшую автономную задачу, принадлежащую проблеме более широкого масштаба.

2. *Категория ToolKit.* Представляет совокупность интегрированных программных средств, обеспечивающих помощь для одного из классов программных задач, использует репозиторий для всей технической и управляющей информации о проекте, концентрируясь при этом на поддержке одной фазы или одного этапа разработки ПО.

3. *Категория WorkBench.* Представляет собой интеграцию программных средств, которые поддерживают системный анализ, проектирование и разработку ПО, используют репозиторий, содержащий всю техническую и управляющую информацию о проекте; обеспечивает автоматическую передачу системной информации между разработчиками и этапом разработки; организует поддержку практически полного жизненного цикла.

Уровни CASE-средств. Классификация по уровням связана с областью действия в пределах ЖЦ ПО. При этом различают три уровня CASE-средств: верхний, средний, нижний.

1. *Верхний (Upper) CASE.* Их часто называют средствами компьютерного планирования, которые призваны повысить эффективность деятельности руководителей фирмы и проекта путем сокращения затрат на определение политики фирмы и на создание общего плана проекта.

2. *Средний (Middle) CASE.* Считаются средствами поддержки этапов анализа требований и проектирования (спецификации и структуры ПО). При использовании этих средств проектирование превращается в итеративный процесс, включающий следующие действия:

– пользователь обсуждает с аналитиком требования к проектируемой системе;

- аналитик документирует эти требования, используя словари входных данных;
- пользователь проверяет эти диаграммы и словари, при необходимости моделирует их;
- аналитик отвечает на эти модификации, применяя соответствующие спецификации.

Эти средства обеспечивают также возможность быстрого документирования требований и прототипирования.

3. *Нижний (Lower) CASE*. Являются средствами разработки ПО. Они содержат системные словари и графику, исключающие необходимость разработки физических спецификаций.

Главным преимуществом этих CASE является: значительное уменьшение времени на разработку; облегчение модификаций; поддержка возможностей прототипирования (совместно со средами CASE).

На современном рынке средств разработки ИС достаточно много систем, в той или иной степени удовлетворяющих требованиям CASE-технологий [4, 16, 17]. Характерной тенденцией является их постоянное совершенствование. Так, фирма Computer Associates (CA) – разработчик известных CASE-средств ERwin, VPwin в 2002 г. выпустила интегральный пакет инструментальных средств, поддерживающих все этапы разработки информационных систем – AllFusion Modeling Suite.

В этот пакет входят пять продуктов:

1. *AllFusion Process Modeler* (новое имя – VPwin) – средство, облегчающее проведение обследования деятельности предприятия и построения функциональных моделей (AS – IS и TO – BE).

2. *AllFusion ERwin Data Modeler* («старое» название – ERwin) – инструмент создания моделей данных и генерации схем баз данных.

3. *AllFusion Data Model Validates* (прежнее название – ERwin Examines) – система поиска и исправления ошибок модели данных.

4. *AllFusion Model Manager* («старое» название – Model Mart) – система организации коллективной работы с хранилищами моделей VPwin, ERwin.

5. *AllFusion Component Manager* («старое» название – Paradigm Plus) – инструмент создания объектных моделей.

В принципе, использовать этот пакет можно, не выходя из него (наподобие мифических гномов, не покидающих лотоса). Кроме того, можно выполнять анализ и проектирование как в случае структурного

системного подхода, так и объектно-ориентированного подхода. В настоящем учебном пособии рассмотрены лишь первые два средства: AllFusion Process Modeler и AllFusion ERwin Data Modeler.

Что касается инструмента поддержки объектно-ориентированной разработки ПО, то в предлагаемом пособии акцентируется внимание на CASE-средстве Rational Rose фирмы Rational Software. Выбор сделан нами скорее с учетом опыта работы и привязанности к этому средству. Последний, конечно, уступает по новизне продукту этой же фирмы Rational XDE для Visual Studio.Net.

Авторы не могли оставить без внимания графический редактор VISIO, который, не являясь CASE-средством, позволяет удовлетворять запросы не только инженеров-механиков, инженеров-электриков, экономистов, хозяйственников, но и разработчиков программного обеспечения. Используя VISIO, можно строить модели программ данных, выполнять операции прямого и обратного инжиниринга. Словом, при отсутствии вышеупомянутых CASE-средств, можно применять VISIO.

Вопросы для самоконтроля

1. Что представляет собой ЖЦ ПО?
2. Назовите традиционно выделяемые этапы ЖЦ ПО.
3. Какие из этих этапов являются критическими и почему?
4. Чем вызвана необходимость использования САПР ПО?
5. Перечислите и охарактеризуйте группы средств автоматизации разработки ПО.
6. Какое место среди этих средств автоматизации занимают CASE-средства?
7. Что представляют собой типы, категории и уровни CASE-средств?
8. Назовите известные вам CASE-средства.

2. МОДЕЛИРОВАНИЕ БИЗНЕС-ПРОЦЕССОВ

2.1. Нотация IDEF0 и моделирование процессов

Диаграммы IDEF0 используются для моделирования широкого класса систем [3–6, 19].

Для новых систем применение IDEF0 имеет своей целью определение требований и указание функций для последующей разработки системы, отвечающей поставленным требованиям и реализующей выделенные функции.

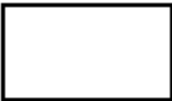
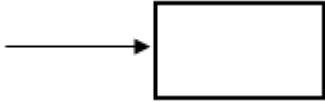
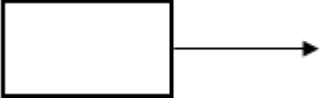
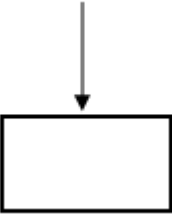
Для существующих систем IDEF0 может быть использована для анализа функций, выполняемых системой и отображения механизмов, посредством которых эти функции реализуются.



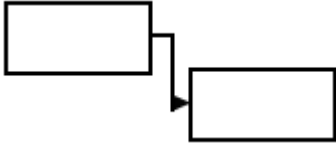
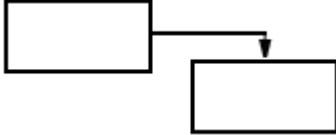
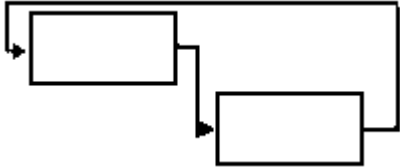
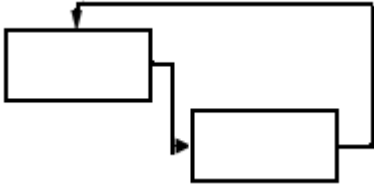
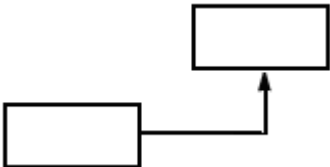
Результатом применения IDEF0 к некоторой системе является модель этой системы, состоящая из иерархически упорядоченного набора диаграмм, текста документации и словарей, связанных друг с другом с помощью перекрестных ссылок.

Основные элементы и понятия, связанные с диаграммой IDEF0, представлены в табл. 2.1.

Таблица 2.1

Основные элементы и понятия, связанные с диаграммой IDEF0

| Наименование | Графическое представление |
|------------------------|--|
| Модуль поведения (UOB) |  |
| Стрелка слева |  |
| Стрелка справа |  |
| Стрелка сверху |  |

| Наименование | Графическое представление |
|--------------------------------------|--|
| Стрелка снизу |  |
| Стрелка вниз |  |
| Типы взаимодействия между процессами | |
| Связь по входу |  |
| Связь по управлению |  |
| Обратная связь по входу |  |
| Обратная связь по управлению |  |
| Связь выход-механизм |  |

Модуль поведения (UOB) служит для описания функций (процедур, работ), выполняемых подразделениями/сотрудниками предприятия.

Стрелка слева описывает входящие документы, информацию, материальные ресурсы, необходимые для выполнения функции.

Стрелка справа характеризует исходящие документы, информацию, материальные ресурсы, являющиеся результатом выполнения функции.

Стрелка сверху описывает управляющие воздействия, например распоряжение, нормативный документ и т. д. В нотации IDEF0 каждая процедура должна обязательно иметь не менее одной стрелки сверху, отражающей управляющее воздействие.

Стрелка снизу характеризует так называемые механизмы, т. е. ресурсы, необходимые для выполнения процедуры, но не изменяющие в процессе ее выполнения свое состояние. Примеры: сотрудник, станок и т. д.

Стрелка вниз изображает связь между разными диаграммами или моделями, указывая на некоторую диаграмму, где данная работа рассмотрена более подробно.

Все работы и стрелки должны быть именованы.

2.2. Инструментальная среда AllFusion Process Modeler

AllFusion Process Modeler является новым именем известного CASE-пакета VPwin. Это CASE-средство предназначено для проведения анализа и реорганизации бизнес-процессов. Оно поддерживает методологии IDEF0 (функциональная модель), IDEF3 (Work Flow Diagram) и DFD (Data Flow Diagram). В AllFusion Process Modeler возможно построение смешанных моделей, т. е. модель может содержать одновременно как диаграмму IDEF0, так и IDEF3 и DFD. При переключении с одной нотации на другую состав палитры компонентов меняется автоматически.

Кроме того, эта инструментальная среда позволяет выполнить стоимостный анализ; задать свойства, определенные пользователем; строить организационные диаграммы и диаграммы Swim Lane.

Ниже в табл. 2.2 рассмотрены операции, выполнение которых не зависит от выбранной нотации моделирования процессов, и рекомендуемая последовательность их осуществления [4–6, 19].

Таблица 2.2

Операции, предшествующие моделированию процессов

| Операция | Рекомендуемая последовательность действий. Примечания |
|----------|--|
|----------|--|

| | |
|--------------------------|---|
| 1. Создание новой модели | { Меню: File → New Основная панель инструментов: [New Model] } → BPwin → [Create model] → Name = <Имя модели> → Type → { IDEF0 IDEF3 DFD } → [OK] |
|--------------------------|---|

Окончание табл. 2.2

| Операция | Рекомендуемая последовательность действий. Примечания |
|---|--|
| 2. Задание свойств новой модели | { [OK] операции 1 Меню: Model → New Model Properties... } → Properties for New Model |
| 3. Задание свойств модели | Меню: Model → New Model Properties... |
| 4. Установка шрифта по умолчанию | Меню: Model → Default Fonts – позволяет установить шрифт по умолчанию для объектов определенного типа на диаграммах в отчетах |
| 5. Отображение всех работ модели в виде раскрывающегося иерархического списка | Model Explorer: <Вкл. Activities> |
| 6. Переход на любую диаграмму модели | Model Explorer: <Вкл. Diagrams> |
| 7. Отображение объектов, диаграммы, выбранной на вкладке Diagrams | Model Explorer: <Вкл. Objects> |
| 8. Открытие существующей модели | { Меню: File → Open Основная панель инструментов: [Open Model] } → <Диалог открыть> |
| 9. Создание контекстной работы | Создается автоматически в выбранной нотации { IDEF0 IDEF3 DFD } при создании новой модели |
| 10. Использование контекстного меню работы | <Работа> → RClick → <Выбор опций: Name... – задание имени работы; Definition/Note... – определение работы и примечания к ней; Font... – изменение шрифта работы; Color – изменение цвета работы; Costs... – задание стоимости работы; Data-Usage... – ассоциация работы с данными; UDP – задание свойств, определяемых пользователем; UOW – задание свойств для работы IDEF3; Box Style – задание особенностей отображения символа работы; Status – задание статуса работы и другие опции> |
| 11. Вызов обучающей программы | Меню: Help → BPwin online Tutorial → Диалог BPwin Tutorial – 10 уроков |

2.3. Разработка IDEF0-моделей процессов в AllFusion Process Modeler

Нотация IDEF0 по замыслу фирмы Computer Associates (разработчика интегрального пакета инструментальных средств AllFusion Modeling Suite) является основной. Она предназначена для описания существующих бизнес-процессов на предприятии (модель AS – IS) и идеального положения вещей – того, к чему следует стремиться (модель TO – BE). Если в процессе моделирования нужно осветить специфические стороны технологического процесса предприятия, возможно переключение на любой ветви модели на нотацию IDEF3 или DFD и создание смешанной модели.

Процесс моделирования системы в IDEF0 начинается с определения наиболее абстрактного уровня его описания (контекста). В контекст входит определение моделируемой системы, цели и точки зрения на модель.

Приступая к моделированию системы, следует определить область (Scope) моделирования. При этом необходимо учитывать два компонента – широту (границы модели) и глубину (уровень детализации завершенной модели).

Цель моделирования (Purpose) должна дать ответ на следующие вопросы:

1. Почему этот процесс должен быть смоделирован?
2. Что должна показывать модель?
3. Что может получить читатель?

Точка зрения (View Point) должна соответствовать цели моделирования и не должна изменяться в процессе моделирования. Как правило, выбирается точка зрения человека (например, руководителя предприятия), ответственного за моделируемую работу в целом. Допускается задокументировать и альтернативные точки зрения на модели. Для этой цели имеется возможность исследовать диаграммы FEO (For Exposition Only).

Технология проектирования ИС подразумевает сначала создание модели AS – IS, ее анализ и изучение бизнес-процессов, т. е. создание модели TO – BE, на основе которой строится модель данных, прототип и затем окончательный вариант ИС.

Ниже приведен список кнопок палитры и инструментов, доступных при работе в нотации IDEF0, а также операции и последовательность их выполнения при выборе упомянутой нотации.

Кнопки палитры инструментов:

- [*Activity Box Tool*] – добавление работы на диаграмму;
- [*Precedence Arrow Tool*] – {новая связь (IDEF0) | стрелка потока данных (DFD) | старшая связь (IDEF3)};
- [*Squiggle Tool*] – создание выносок к именам стрелок;
- [*Text Tool*] – внесение текста в окно диаграммы;
- [*Diagram Dictionary Editor*] – переход в редактор словаря;
- [*Go to Sibling Diagram*] – переключение между диаграммами;
- [*Go to Parent Diagram*] – переход на диаграмму верхнего уровня;
- [*Go to Child Diagram*] – переход на диаграмму нижнего уровня, декомпозиция диаграммы;
- [*Pointer Tool*] – редактирование объектов.

Операции, выполняемые в нотации IDEF0, и рекомендуемая последовательность их осуществления даны в табл. 2.3.

Таблица 2.3

Операции, выполняемые в нотации IDEF0

| Операция | Рекомендуемая последовательность действий. Примечания |
|--|--|
| 1. Внесение граничной стрелки, входящей в работу | Палитра инструментов: [<i>Precedente Arrow Tool</i>] → 1{<{Левая Нижняя Верхняя} сторона экрана> → Click → <Сторона входа стрелки> → Click}* – повторить для всех стрелок, входящих в работу → [<i>Pointer Tool</i>] |
| 2. Внесение граничной стрелки, исходящей от работы | <Палитра инструментов> → [<i>Precedente Arrow Tool</i>] → 1{<Правая сторона работы> → Click → <Правая сторона экрана> → Click}* – повторить для всех входящих стрелок → [<i>Pointer Tool</i>] |
| 3. Использование контекстного меню стрелки | <Стрелка> → RClick → <Name... – задать имя стрелки; то же самое, что и для работы, только для стрелки; Squiggle – создать/убрать выноску к имени стрелки; Trim – укоротить стрелку; Arrow Tunnel – операции тоннелирования стрелок и другие опции> |
| 4. Декомпозиция контекстной диаграммы (в рамках нотации IDEF0) включает этапы: | |

| | |
|--|--|
| 4.1. Декомпозиция контекстной работы | <Контекстная работа> → Click → Палитра инструментов: [Go to Child Diagram] → <Activity Box Count> → Number of Activities in this = <Число работ> → [OK] – в окне новой диаграммы появятся заготовки работ декомпозиции и мигрировавшие граничные стрелки |
| 4.2. Задание свойств работ декомпозиции | 1{<Одна из работ> → RClick → <Выбор требуемых опций>}* – повторить для всех работ декомпозиции |
| 4.3. Связывание граничных стрелок с работами | 1{<Одна из стрелок> → Click_∨ → <Требуемое место> → Click_∧ → <Конец стрелки> → Click_∨ → <Сторона работы> → Click_∧ – для разветвлений стрелки использовать [Precedence Arrow Tool]}* – повторить для всех стрелок |

Окончание табл. 2.3

| Операция | Рекомендуемая последовательность действий. Примечания |
|---|--|
| 4.4. Создание внутренних стрелок | 1{Палитра инструментов: [Precedence Arrow Tool] → <Работа-источник> → Click → <Работа-приемник> → Click} – повторить для всех стрелок → Палитра инструментов: [Pointer Tool] |
| 4.5. Задание свойств внутренних стрелок | 1{<Одна из стрелок> → RClick → <Выбор требуемых опций>}* – повторить для всех внутренних стрелок |
| 4.6. Создание новой граничной стрелки (при необходимости) | <Сторона требуемой работы> – строится граничная стрелка, которая не попадает на верхний уровень и помечается квадратными скобками [] |
| 4.7. Тоннелирование стрелок | <Конец стрелки []> → RClick → Arrow Tunnel → {[Resolve it to borderarrow] – стрелка попадает на верхний уровень [Change it to resolved rounded tunnel] – стрелка будет затоннелирована круглым тоннелем ()} → [OK] |

Построение IDEF0-фрагмента модели AS – IS при разработке ПО «Информационная система обработки полиграфических заказов».

Система предназначена для принятия и обработки заказов в полиграфической структуре. В зависимости от требований заказчиков работники (администратор, оператор и т. д.) типографии сформировали базу данных, в которой отображены типы операций на различном оборудовании, учитывающие все запросы клиентов.

Основным назначением системы является автоматизация ввода и хранения данных по использованию полиграфического оборудования и различных типов печати. Система позволяет изменять, дополнять,

вести поиск и просмотр информации о заказах. Также система адекватно распределяет заказы по типу требований во всей полиграфической системе, что ограничивает заказы от временных задержек на время выполнения требуемой работы.

Технология проектирования информационных систем подразумевает создание сначала модели AS – IS, ее анализ и улучшение бизнес-процессов, т. е. создание модели TO – BE.

На основе последней строится модель данных, прототип и окончательный вариант информационной системы.

Для создания упомянутых моделей рекомендуется выполнить консалтинг деятельности организации-заказчика, включающий следующую последовательность работ.

1. Проведение функционального и информационного обследования целевой деятельности:

- определение оргштатной и топологической структур организации;
- установление перечня целевых задач (функций) организации;
- анализ распределения функций по подразделениям и сотрудникам;
- формирование альбома форм входных и выходных документов, используемых организацией.

2. Разработка структурной функциональной модели деятельности организации:

- определение информационных потоков между основными процессами деятельности, связи между процессами и внешними объектами;
- оценка объемов и интенсивности информационных потоков;
- разработка иерархии диаграмм потоков данных, обращающих структурную функциональную модель деятельности;
- анализ и оптимизация структурной физической модели.

3. Разработка информационной модели организации:

- определение сущностей моделей и их атрибутов;
- проведение атрибутивного анализа и оптимизация сущностей;
- идентификация отношений и определение типов отношений;
- разрешение неспецифических отношений (многие ко многим);
- анализ и оптимизация информационной модели.

4. Разработка событийной модели организации:

- идентификация перечня состояний модели и определение возможных переходов между состояниями;
- определение условий, активизирующих переходы, и действий, влияющих на дальнейшее поведение;

– анализ и оптимизация событийных моделей.

5. Разработка предложений по автоматизации организации:

– составление перечня автоматизированных рабочих мест (АРМ) и способов взаимодействия между ними;

– подготовка требований к техническим средствам;

– исследование требований к программным средствам;

– разработка предложений по средствам взаимодействия подразделений;

– исследование предложений по этапам и срокам автоматизации.

Таким образом, фактически строится два типа моделей.

1. *Модель деятельности (AS – IS)*, представляющая собой «снимок» положения дел в организации на момент обследования и позволяющая понять, что делает и как функционирует организация с позиций системного анализа, а также на основании автоматической верификации выявить ряд ошибок и узких мест и сформулировать предложения по улучшению ситуации.

2. *Модель автоматизации (TO – BE)* интегрирует перспективные предложения руководства и сотрудников организации, экспертов и системных аналитиков и позволяет сформировать видение новой (автоматизированной) системы, а именно: что вновь создаваемая система будет делать и как она будет функционировать.

При построении модели AS – IS используют результаты обследования деятельности организации и создают контекстную диаграмму. На рис. 2.1 показано обслуживание клиента в полиграфической системе (ПС). В ПС поступают заказы от клиентов. Администратор принимает заказы и формирует из них очередь. После обработки ПС сформированных заказов создается откорректированная информация о заказах и внедряются полученные результаты заказов.

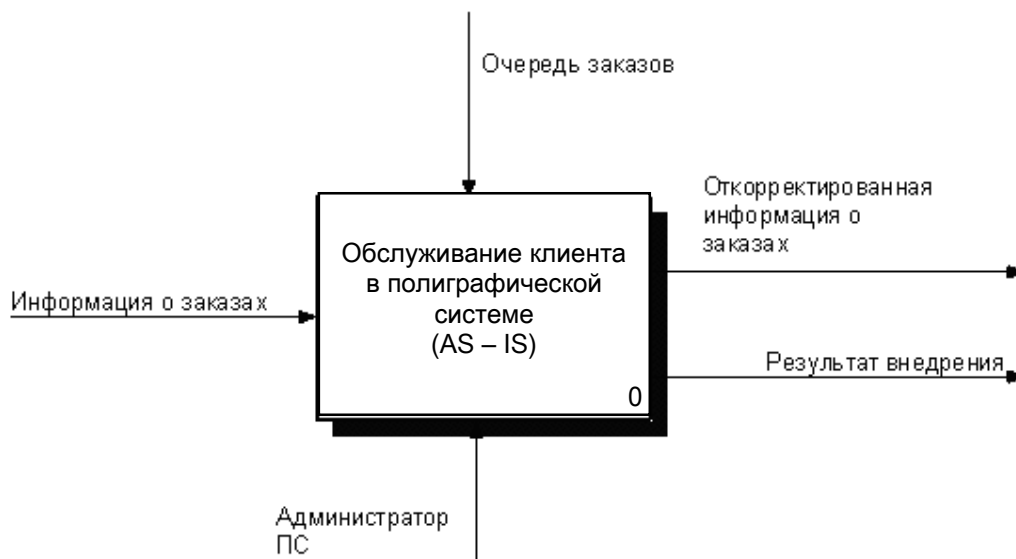


Рис. 2.1. Контекстная диаграмма процесса «Обслуживание клиента в полиграфической системе»

Далее выполняют декомпозицию процесса «Обслуживание клиента в полиграфической системе» на три процесса: «Подготовка рекламной акции», «Обращение в ПС», «Внедрение выполненного заказа в рекламную акцию» (рис. 2.2). Исполнителем процесса «Подготовка рекламной акции» является администратор ПС. В результате выполнения процесса на выходе получают информацию о заказе клиента, которая затем передается в качестве управляющей процесса «Обращение в ПС».

Вследствие реализации процесса «Обращение в ПС», исполнителем которого является администратор ПС, получают на выходе два потока: «Откорректированная информация о заказах» и «Представление выполненного заказа в ПС офсетной печати». Последний передается в качестве управляющего в процесс «Внедрение выполненного заказа в рекламную акцию». На вход рассматриваемого процесса поступает информация о заказах. Управляют этим процессом два потока: «Очередь заказов» и «Заказ клиента».

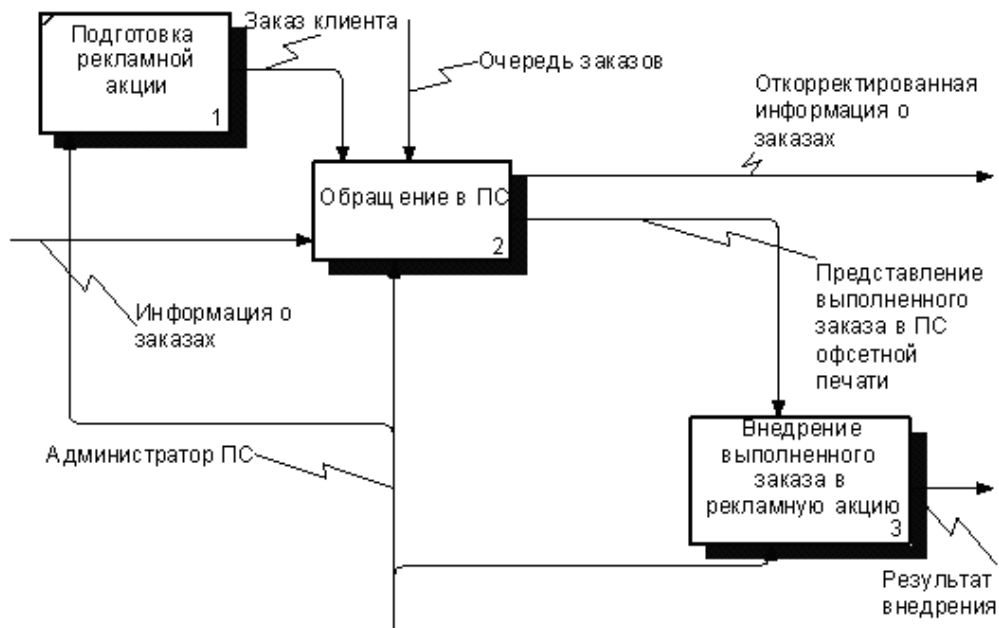


Рис. 2.2. Декомпозиция процесса «Обслуживание клиента в полиграфической системе»

В результате выполнения процесса «Внедрение выполненного заказа в рекламную акцию», исполнителем которого является администратор, получают поток «Результат внедрения». Управляет этим процессом поток «Представление выполненного заказа».

Затем осуществляют декомпозицию процесса «Обращение в PC» на три процесса: «Внесение заказа в картотеку», «Выбор печати», «Выполнение заказа» (рис. 2.3). Управляет процессом «Внесение заказа в картотеку» два потока: «Очередь заказов», «Заказ клиента». На вход указанного процесса подается информация о заказах. В результате исполнителем процесса «Внесение заказа в картотеку» является администратор ПС. В итоге выполнения этого процесса на выходе получают два потока: «Откорректированная информация о заказах» и «Выполнение заказа», который в свою очередь является управлением процесса «Выбор печати».

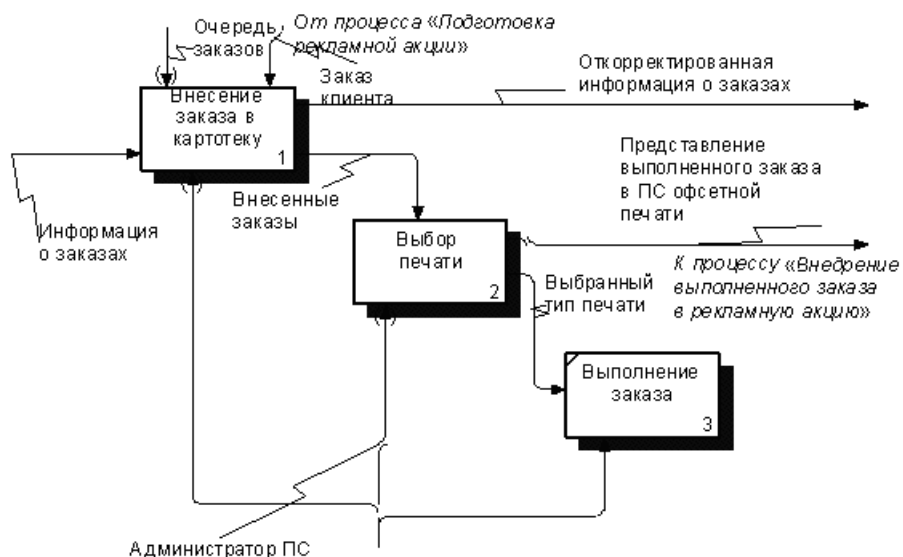


Рис. 2.3. Декомпозиция процесса «Обращение в полиграфическую систему»

В ходе реализации процесса «Выбор печати», исполнителем которого является администратор ПС, на выходе получают два потока: «Представление выполненного заказа в ПС офсетной печати», который затем переходит к управлению процессом «Внедрение выполненного заказа в рекламную акцию», и «Выбранный тип печати». Поток «Выбранный тип печати» является управляющим процесса «Выполнение заказа». Исполнителем процесса «Выполнение заказа» выступает администратор ПС.

2.4. Редактор VISIO и нотация IDEF0

Графический редактор VISIO предназначен для разработки различных графических приложений и документов. Это простой и в то же время мощный инструмент для создания:

- блок-схем, карт компании, маркетинговых диаграмм;
- диаграмм поддержки структурного системного анализа и проектирования при разработке ПО;
- диаграмм объектно-ориентированного анализа и проектирования при разработке ПО.

VISIO находит широкое применение в работе экономистов, администраторов, инженеров-программистов, инженеров-механиков, инженеров-электриков и многих других специалистов. Особенно ценен этот пакет при создании презентаций. Причем встроенный в него Visual Basic for Application позволяет создавать прекрасные VISIO-приложения.

В ряде случаев VISIO не уступает по своим возможностям таким известным CASE-пакетам, как BPwin, ERwin, Rational Rose, а кое-где превосходит их. Что касается многоцелевого его использования, то ему нет равных. В литературе, правда, можно найти различные точки зрения на VISIO. Все, пожалуй, зависят от того, насколько тот или иной пользователь использует VISIO и владеет его возможностями.

Поскольку применение VISIO не является объектом рассмотрения данного учебного пособия, постараемся весьма кратко, но объективно отличить возможности этого редактора в части, где он может быть альтернативой рассматриваемой здесь САПР ПО. В частности, при отсутствии таких CASE-средств, как AllFusion Process Modeler, AllFusion ERwin Data Modeler, Rational Rose, моделирование бизнес-процессов, баз данных и разрабатываемого программного обеспечения можно выполнить, прибегнув к услугам VISIO.

VISIO как инструмент построения IDEF0-моделей ПО все же уступает CASE-средству AllFusion Process Modeler. Это объясняется тем, что предшественники последнего – пакеты семейства BPwin были изначально ориентированы на поддержку нотации IDEF0 в качестве основной. Отсюда и высокое качество сервиса, предоставляемого бизнес-аналитиком при построении IDEF0-моделей в среде AllFusion Process Modeler. Тем не менее, выбрав в редакторе VISIO Меню: File → Stencils → Flowchart → IDEF0 Diagram Shapes, можно воспользоваться появившемся трафаретом, показанным на рис. 2.4, для построения диаграмм IDEF0.

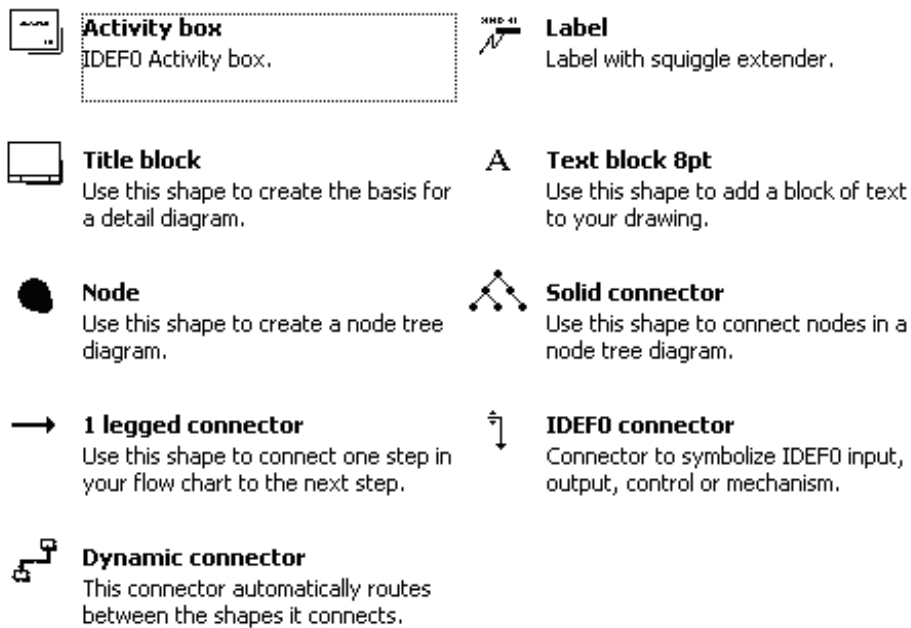


Рис. 2.4. Общий вид трафарета IDEF0 «Diagram Shapes»

Вопросы для самоконтроля

1. Для чего предназначена модель в нотации IDEF0?
2. Что обозначают работы в IDEF0?
3. Каково назначение сторон прямоугольников работ на диаграммах IDEF0?
4. Перечислите типы стрелок, применяемых на диаграммах IDEF0.
5. Какой инструментарий используется для построения диаграмм IDEF0?
6. Как осуществляется построение алгоритма в среде VISIO?
7. Формирование структуры баз данных в среде VISIO.

3. НОТАЦИЯ DFD И МОДЕЛИРОВАНИЕ ПРОЦЕССОВ

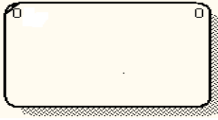



3.1. Элементы нотации DFD

Диаграммы потоков данных (Data Flow Diagram [1, 2, 5, 6, 19]) применяются для моделирования функциональных требований проектируемой системы и позволяют продемонстрировать, каким образом каждый процесс преобразует свои входные данные в выходные, а также выявить отношения между процессами.

Основные элементы и понятия, связанные с диаграммой потоков данных, представлены в табл. 3.1.

Таблица 3.1

Основные элементы диаграммы потоков данных

| Наименование | Графическое представление |
|-------------------------------------|--|
| Работа (Activity) |  |
| Информационный поток (Precedence) |  |
| Внешняя ссылка (External reference) |  |
| Хранилище данных (Data store) |  |

Работа (Activity) описывает функции или процессы, которые обрабатывают и изменяют информацию.

Информационный поток (Precedence) обозначает информационный поток от объекта-источника к объекту-приемнику.

Внешняя ссылка (External reference) указывает на место, организацию или человека, которые участвуют в процессе обмена информацией с системой, но располагаются за рамками этой диаграммы.

Хранилища данных (Data store) представляют собой собственно данные, к которым осуществляется доступ. Эти данные также могут быть созданы или изменены работами. На одной диаграмме может

присутствовать несколько копий одного и того же хранилища данных.

3.2. Разработка DFD-моделей процессов в AllFusion Process Modeler

Нотация DFD в AllFusion Process Modeler является дополнением к IDEF0. Как уже упоминалось выше, переключение с нотации IDEF0 на DFD возможно на любой ветви модели. Это по замыслу разработчиков AllFusion Process Modeler позволяет более полно показать моделируемый процесс. В AllFusion Process Modeler для построения используется нотация Гейка – Карсона. При этом допускается некоторое отклонение от синтаксиса и применение в DF-диаграммах отдельных сообщений построения моделей в IDEF0. В связи с чем при переключении с IDEF0 на DFD прежняя палитра инструментов лишь дополняется необходимыми для работы кнопками. С нотации DFD можно на любой ветви модели переключиться на нотацию IDEF3. Можно начинать моделирование процессов сразу и с нотации DFD, трактуя ее как базовую.

Ниже приведен список кнопок палитры и инструментов, доступных при работе в нотации DFD, а также операции и последовательность их выполнения при выборе упомянутой нотации.

Дополнительные кнопки палитры инструментов:

[*External Reference*] – добавление в диаграмму внешней ссылки;

[*Data Store*] – добавление в диаграмму хранилища данных.

Поскольку выполнение многих операций, связанных с построением DF-диаграмм, мало отличается от рассмотренных при моделировании процессов в нотации IDEF0, в табл. 3.2 приведены лишь этапы декомпозиции IDEF0-процесса в DF-диаграмму и создания межстрочной ссылки [4–6, 19].

Таблица 3.2

Операции декомпозиции IDEF0-процесса в DF-диаграмму

| Операция | Рекомендуемая последовательность действий. Примечания |
|---|--|
| 1. Декомпозиция IDEF0-процесса в DF-диаграмму 1.1. Декомпозиция IDEF0-работы | <Декомпозируемая работа> → Click → Палитра инструментов: [Go to Child Diagram] → Activity Box Count → [DFD] → Number of Activities in this = |

| | |
|--|---|
| | <Число работ> – в окне новой диаграммы появятся заготовки работ декомпозиции и мигрировавшие граничные стрелки, которые следует удалить |
|--|---|

Окончание табл. 3.2

| Операция | Рекомендуемая последовательность действий. Примечания |
|--|--|
| 1.2. Задание свойств работ декомпозиции | 1{<Одна из работ> → RClick → <Выбор требуемых опций>}* – повторить для всех работ |
| 1.3. Создание внешних сущностей и хранилищ | {Палитра инструментов: {[External Reference] [Data Store]} → <Место в окне диаграммы> → Click = {<Имя хранилища> <Имя внешней ссылки>}* – повторить для всех внешних ссылок либо хранилищ |
| 1.4. Создание внутренних стрелок | Как в случае IDEF0-нотации. Только здесь стрелки строятся не только между работами, но и хранилищами и работами, внешними ссылками и работами |
| 1.5. Тоннелирование стрелок | <Декомпозируемая работа> – тоннелируются все стрелки, входящие и исходящие из работы |
| 2. Внесение межстраничной ссылки | Палитра инструментов → [Precedence Arrow Tool] → <Окно диаграммы> → <Требуемая работа> → Click → <Правая сторона экрана> → Click → <Наконечник стрелки> → RClick → Off-Page Arrow Reference → <Diagram> = <Выбранная диаграмма> → Меню: Model → Model Properties → Display → Off-Page Reference – Node number → <Диаграмма-приемник> → <Наконечник появившейся граничной стрелки> → Click → <Требуемая работа> → Click |

Продолжение создания модели AS – IS при разработке ПО «Информационная система обработки полиграфических заказов». При осуществлении декомпозиции процесса «Внесение заказа в картотеку» DF-диаграмма разбивается на три процесса: «Внести заказ в основную картотеку ПС», «Принять заказ или отклонить», «Сортировка в картотеке заказов» (рис. 3.1).

В результате выполнения процесса «Внести заказ в основную картотеку ПС» производится внесение заказа в основную картотеку ПС. Входными потоками процесса являются «Заказ клиента» и

«Очередь заказов», извлекаемая из неоткорректированной картотеки, представленной на диаграмме в виде хранилища 1. Выходом из процесса считается поток «Откорректированная информация о заказах», которая подается на вход процесса «Принять заказ или отклонить».

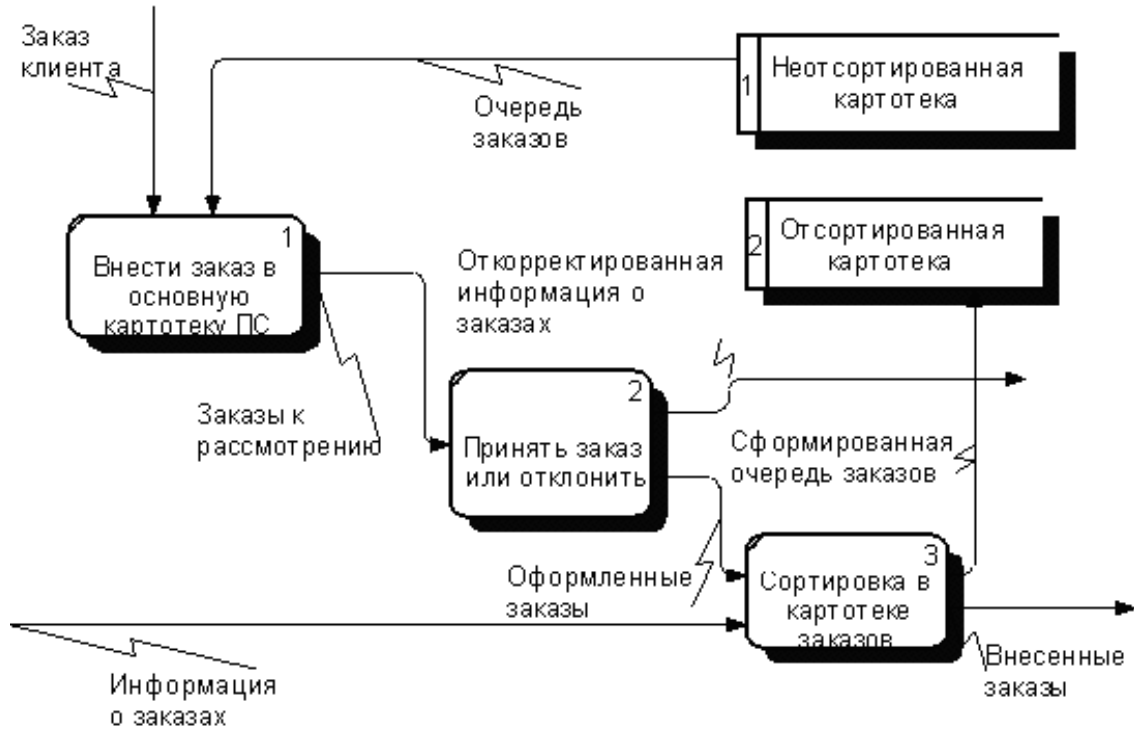


Рис. 3.1. Декомпозиция процесса «Внесение заказа в картотеку» в DF-диаграмму

В ходе реализации процесса «Принять заказ или отклонить» получают на выходе два потока: «Откорректированная информация о заказах» и «Оформленные заказы». Последний подается на процесс «Сортировка в картотеке заказов».

Выполняя процесс «Сортировка в картотеке заказов», получают два потока: «Внесенные заказы» и «Сформированная очередь заказов». Последний помещается в отсортированную картотеку. На вход рассматриваемого процесса, помимо потока «Оформленные заказы», подается поток «Информация о заказах».

При осуществлении декомпозиции процесса «Выбор печати» DF-диаграмма разбивается на три процесса: «Выбор типа и формы печати заказа», «Выбор печатного оборудования», «Выполнение заказа офсетной печатью» (рис. 3.2).

В ходе выполнения процесса «Выбор типа и формы печати заказа» заказ вносится в картотеку заказов клиента. Входными потоками процесса являются «Внесенные заказы» и «Принятые заказы». Последние извлекаются из картотеки заказов клиента. Выходом из рассматриваемого процесса считается поток «К выбору печатного оборудования».

Вследствие реализации процесса «Выбор печатного оборудования» по входному потоку «К выбору печатного оборудования» получают два выходных потока: «Выполнение заказа» и «Заказ в печать». Последний является входом процесса «Выполнение заказа офсетной печатью».

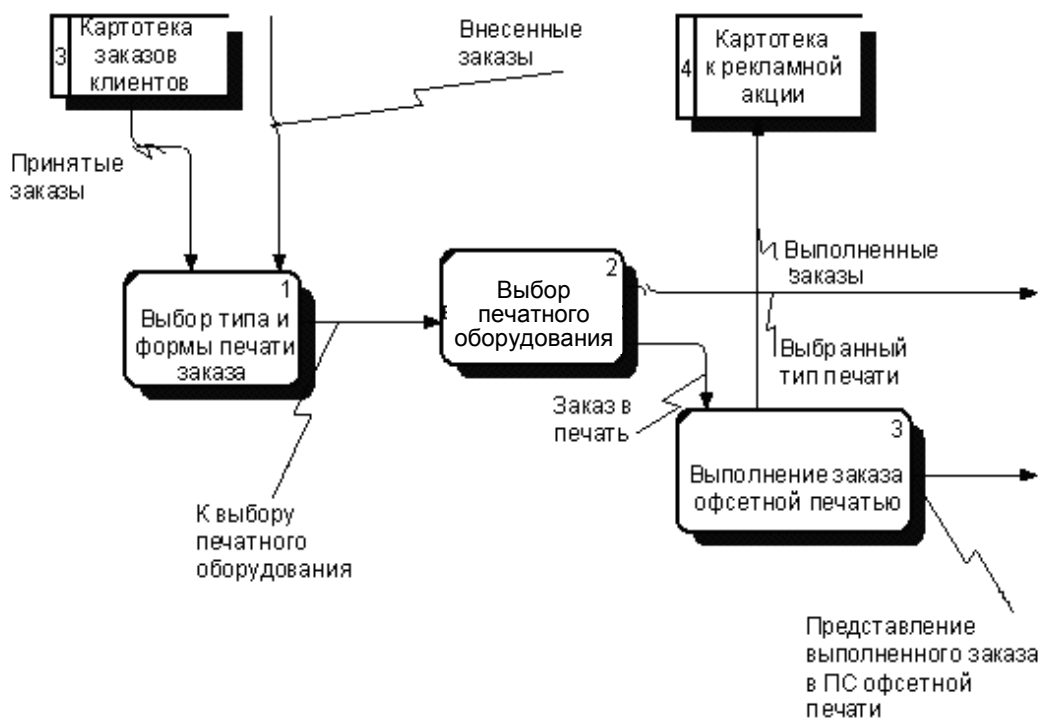


Рис. 3.2. Декомпозиция процесса «Выбор печати» в DF-диаграмму

Результатом реализации процесса «Выполнение заказа офсетной печатью» по входному потоку «Заказ в печать» являются два потока: «Представление выполненного заказа в ПС офсетной печати» и «Выбранный тип печати». Последний помещается в картотеку к рекламной акции.

3.3. Редактор VISIO и нотация DFD

VISIO как инструмент построения DF-диаграмм превосходит средство AllFusion Process Modeler, предшественником которого является пакет семейства VPwin. DFD-нотация была подключена по непланируемой изначально необходимости. Отсюда и отсутствие в AllFusion Process Modeler строгой поддержки синтаксиса и нотации для создания этого класса диаграмм.

Другое дело графический редактор VISIO. Выбрав в редакторе VISIO Меню: File → Stencils → Flowchart → Data Flow Diagram Shapes, можно воспользоваться появившимся трафаретом (рис. 3.3) для построения DF-диаграмм в нотации Йодана, а выбрав Меню: File → Stencils → Software → Gane-Sarson, – появившимся трафаретом (рис. 3.4) для построения DF-диаграмм в нотации Гейка – Сарсона.

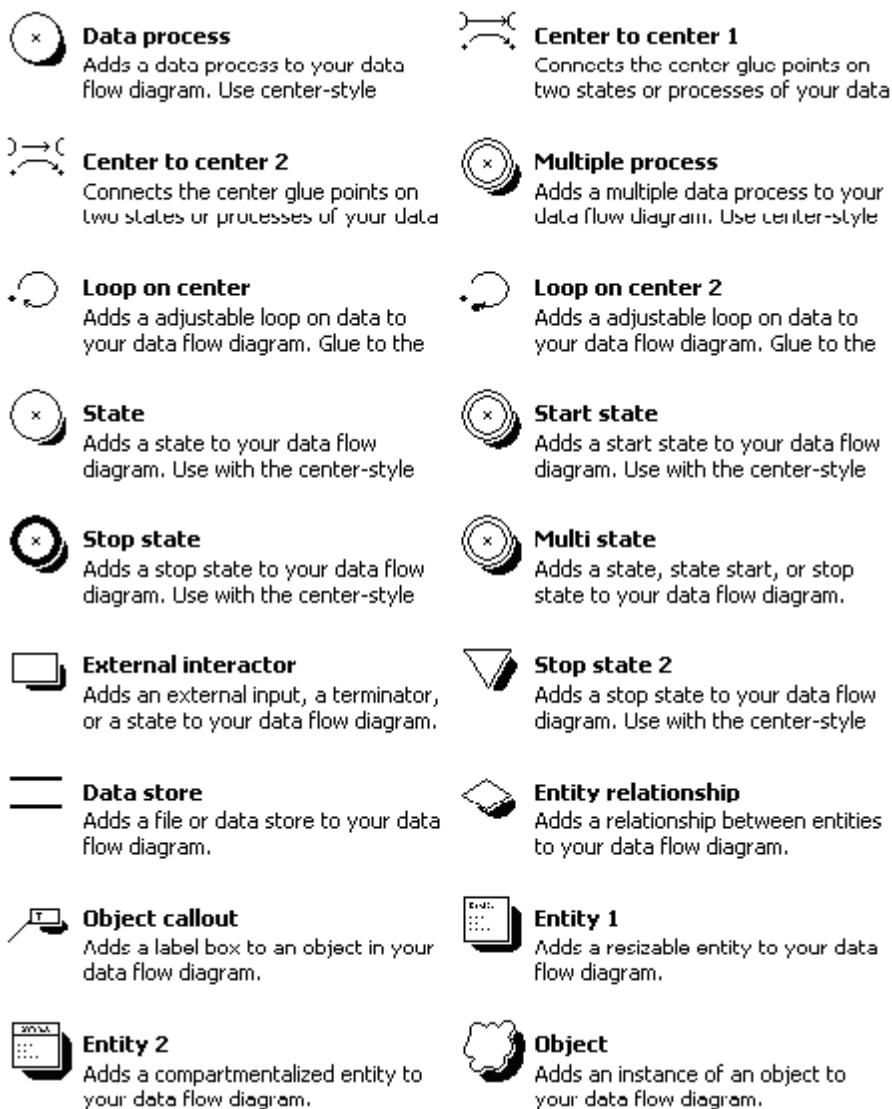


Рис. 3.3. Общий вид трафарета с раздела Flowchart для поддержки построения DF-диаграмм в нотации Иодана

При необходимости несколько модифицировать значки или ввести новые, как это сделано в CASE-пакете 1-го уровня генерации CASE-аналитика, можно исходный значок процесса нотации Гейка – Сарсона перестроить в трехсекционный (1-я секция для записи номера, 2-я – имени процесса, 3-я – исполнителя), а также повернуть последний на 90° и растянуть по горизонтали, дополнив арсенал значков знаком магистрали.

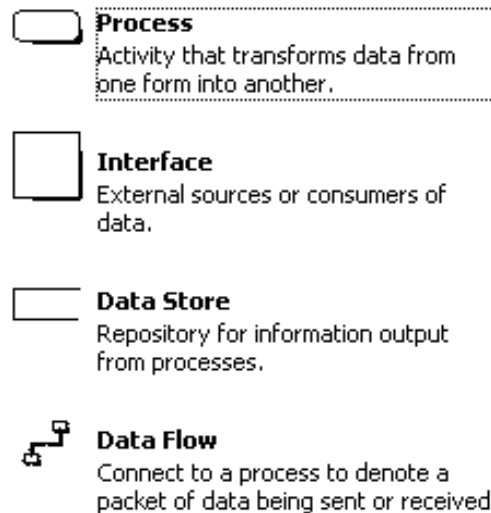


Рис. 3.4. Общий вид трафарета с раздела Software для поддержки построения DF-диаграмм в нотации Гейка – Сарсона

Вопросы для самоконтроля

1. Для чего предназначена DFD-диаграмма?
2. Перечислите основные элементы DFD-диаграммы.
3. В чем состоит отличие диаграммы IDEF0 от диаграммы DFD?
4. Какая нотация используется в VPwin для построения DFD-диаграммы?
5. Как дополнить диаграмму IDEF0 диаграммой DFD?
6. Какие нотации DFD-диаграмм поддерживают графический редактор VISIO?

4. НОТАЦИЯ IDEF3 И МОДЕЛИРОВАНИЕ ПРОЦЕССОВ

4.1. Элементы нотации IDEF3

Диаграммы IDEF3 применяются для более удобного описания рабочих процессов (workflow), для которых важно отразить логическую последовательность выполнения процедур [3–6].

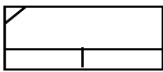
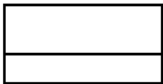



Наличие в диаграммах DFD элементов для описания источников, приемников и хранилищ данных позволяет точно описать процесс документооборота. Однако для описания логики взаимодействия информационных потоков модель дополняют диаграммами еще одной методологии – IDEF3, также называемой workflow diagramming. Методология моделирования IDEF3 дает возможность графически описать и задокументировать процессы, фокусируя внимание на течении этих процессов и на отношениях процессов и важных объектов, являющихся частями этих процессов.

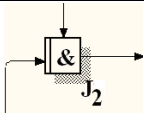
IDEF3 предполагает построение двух типов моделей: модель может отражать некоторые процессы в их логической последовательности, позволяя увидеть, как функционирует организация, или же модель может показывать «сеть переходных состояний объекта», предлагая вниманию аналитика последовательность состояний, в которых может оказаться объект при прохождении через определенный процесс.

Основные элементы и понятия, связанные с диаграммой IDEF3, представлены в табл. 4.1.

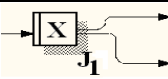



Таблица 4.1

Основные элементы диаграммы IDEF3

| Наименование | Графическое представление |
|------------------------------------|---|
| Единица работы (Unit of Work) |  |
| Объект ссылки (Referents) |  |
| Связь предшествования (Precedence) |  |
| Связь отношения (Relational) |  |
| Поток объектов (Object Flow) |  |

| | |
|---------------------------------------|---|
| Перекресток слияния (Fan-in Junction) |  |
|---------------------------------------|---|

Окончание табл. 4.1

| Наименование | Графическое представление |
|--|---|
| Перекресток ветвления (Fan-out Junction) |  |
| Логическое «И» |  |
| Логическое «ИЛИ» |  |
| Логическое исключающее «ИЛИ» |  |

Единица работы (Unit of Work) служит для описания функций (процедур, работ), выполняемых подразделениями/сотрудниками предприятия.

Объект ссылки (Referents) используется для описания ссылок на другие диаграммы модели, циклические переходы в рамках одной модели, различные комментарии к функциям.

Связи (Links) – связи, изображаемые стрелками, отражают взаимоотношения работ. В IDEF3 различают три типа связей.

Связь предшествования (Precedence) показывает, что прежде чем начнется работа-приемник, должна завершиться работа-источник. Обозначается сплошной линией.

Связь отношения (Relational) отражает связь между двумя работами или между работой и объектом ссылки. Обозначается пунктирной линией.

Поток объектов (Object Flow) показывает участие некоторого объекта в двух или более работах, как например, если объект производится в ходе выполнения одной работы и потребляется другой работой. Обозначается стрелкой с двумя наконечниками.

Перекрестки (Junctions) используются в диаграммах IDEF3, чтобы показать ветвления логической схемы моделируемого процесса и альтернативные пути развития процесса, которые могут возникнуть во время его выполнения.

Перекресток слияния (Fan-in Junction) – узел, собирающий множество стрелок в одну, указывая на необходимость условия завершения работ-источников стрелок для продолжения процесса.

Перекресток ветвления (*Fan-out Junction*) – узел, в котором единственная входящая в него стрелка ветвится, показывая, что работы, следующие за перекрестком, выполняются параллельно или альтернативно.

Логическое «И», логическое «ИЛИ», логическое исключаящее «ИЛИ» – логические операторы, определяющие связи между функциями в рамках процесса. Дают возможность описать ветвление процесса.

4.2. Разработка IDEF3-моделей процессов в среде AllFusion Process Modeler

Нотация IDEF3 в AllFusion Process Modeler является дополнением к IDEF0. Как уже упоминалось выше, переключение с нотации IDEF0 либо DFD на IDEF3 возможно на любой ветви модели. Это позволяет более многогранно представить объект моделирования. AllFusion Process Modeler допускает некоторое отклонение от синтаксиса IDEF3, так как палитра инструментов при переходе с IDEF0 на IDEF3 лишь дополняется кнопками необходимых при этом инструментов. Можно начинать моделирование процессов сразу и с нотации IDEF3. Однако переключение с IDEF3 на другие нотации невозможно.

Ниже приведен список дополнительных кнопок палитры инструментов, появляющийся при переключении с нотации IDEF0 на нотацию IDEF3, а также операции и последовательность их выполнения при выборе рассматриваемой нотации.

Дополнительные кнопки палитры инструментов:

[Referent] – добавление объекта ссылки;

[Junction] – добавление перекрестка.

Поскольку в этом разделе имеет смысл акцентировать внимание на операциях, связанных с декомпозицией IDEF0- либо DF-работы в диаграмму IDEF3 и слиянием/расщеплением модели, данные операции и последовательность их выполнения при выборе рассматриваемой нотации представлены в табл. 4.2 [5, 6, 19].

Таблица 4.2

Операции, связанные с декомпозицией IDEF0- либо DF-работы в диаграмму IDEF3 и слиянием/расщеплением модели

| Операция | Рекомендуемая последовательность действий. Примечания |
|--|--|
| 1. Декомпозиция IDEF0- либо DFD-работы в диаграмму нотации IDEF3 | |

Окончание табл. 4.2

| Операция | Рекомендуемая последовательность действий. Примечания |
|---|---|
| 1.1. Декомпозиция IDEF0- либо DFD-работы | <Декомпозируемая работа> → Click → Палитра инструментов: [Go to Child Diagram] → Activity Box Count → [IDEF3] → Activity in this = <Число работ> – в окне новой диаграммы появятся заготовки работ декомпозиции |
| 1.2. Задание свойств работ декомпозиции | Как и в случае других нотаций, используются опции контекстных диаграмм работ |
| 1.3. Внесение перекрестков в диаграмму | Палитра инструментов: [Junction] → 1{<Требуемое место окна диаграммы> → Click → Select Junction Type = <Выбранный тип> → Click}* – повторить для всех перекрестков |
| 1.4. Добавление объекта ссылки | <Палитра инструментов> → [Referent] → 1{<Требуемая область окна диаграммы> → Click = <Имя объекта ссылки> → [OK]}* – повторить для всех объектов |
| 1.5. Задание свойств объекта ссылки | Используются опции контекстного меню ссылки |
| 1.6. Задание связей между объектами диаграммы | Как и в случае других нотаций, но соблюдая правила создания перекрестков во избежание конфликтов |
| 2. Расщепление модели | Окно диаграммы: <Работа-предок отщепляемой ветви> → RClick → Split Option → Name for new model = <Имя новой модели> → [Copy entire dictionaries] – появится новая модель, а на работе-предке – стрелка вызова с именем новой модели |
| 3. Слияние модели | <Работа со стрелкой вызова> → RClick → Merge Model → [Cut/Paste entire dictionaries] → [OK] – стрелка вызова на работе-предке исчезла, модели слились, но копия подключаемой модели осталась и может быть сохранена |

Продолжение создания модели AS – IS при разработке ПО «Информационная система обработки полиграфических заказов». При декомпозиции процесса «Выполнение заказа» IDEF3-диаграмма включает в себя пять работ: «Обработка заявок в полиграфии», «Оформление документации по заказу», «Дооформление полиграфического заказа», «Формирование выполненного печатного заказа» и «Составление отчетности» (рис. 4.1).

Работа «Обработка заявок в полиграфии» требует выполнения одной из двух работ – «Оформление документации по заказу», либо «Дооформление полиграфического заказа». Работа «Оформление документации по заказу» передает управление на две параллельные работы: «Формирование выполненного печатного заказа» и «Составление отчетности».

Причем работа «Обработка заявок в полиграфии» обращается к двум ссылочным объектам: «Заказы клиентов» и «Картотека полиграфической системы». А работа «Дооформление полиграфического заказа» – к ссылочному объекту «Клиент». Работа «Формирование выполненного печатного заказа» – к ссылочному объекту «Заказы клиентов».

Перекресток J_1 позволяет реализовать работу «Оформление документации по заказу» либо работу «Дооформление полиграфического заказа». Работа «Оформление документации по заказу» передает управление на две параллельные работы: «Формирование выполненного печатного заказа» и «Составление отчетности».

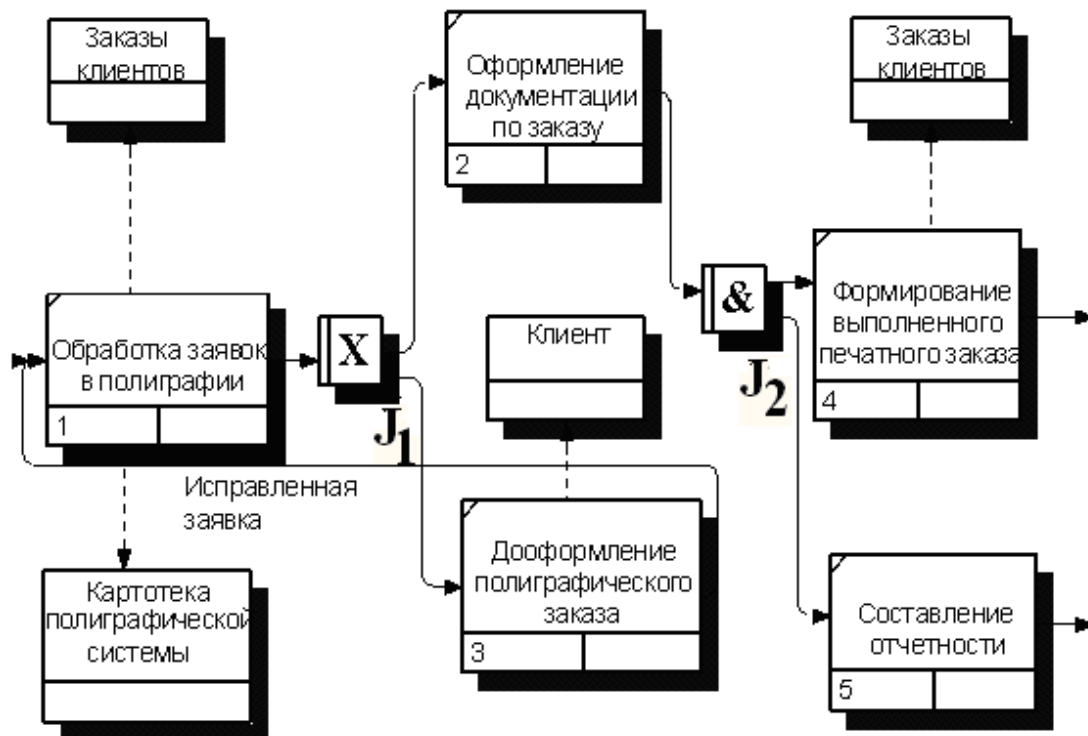


Рис. 4.1. Декомпозиция процесса «Выполнение заказа» в IDEF3-диаграмму

В завершение создания модели AS – IS строится диаграмма дерева узлов, которая может быть представлена в виде, показанном на рис. 4.2.

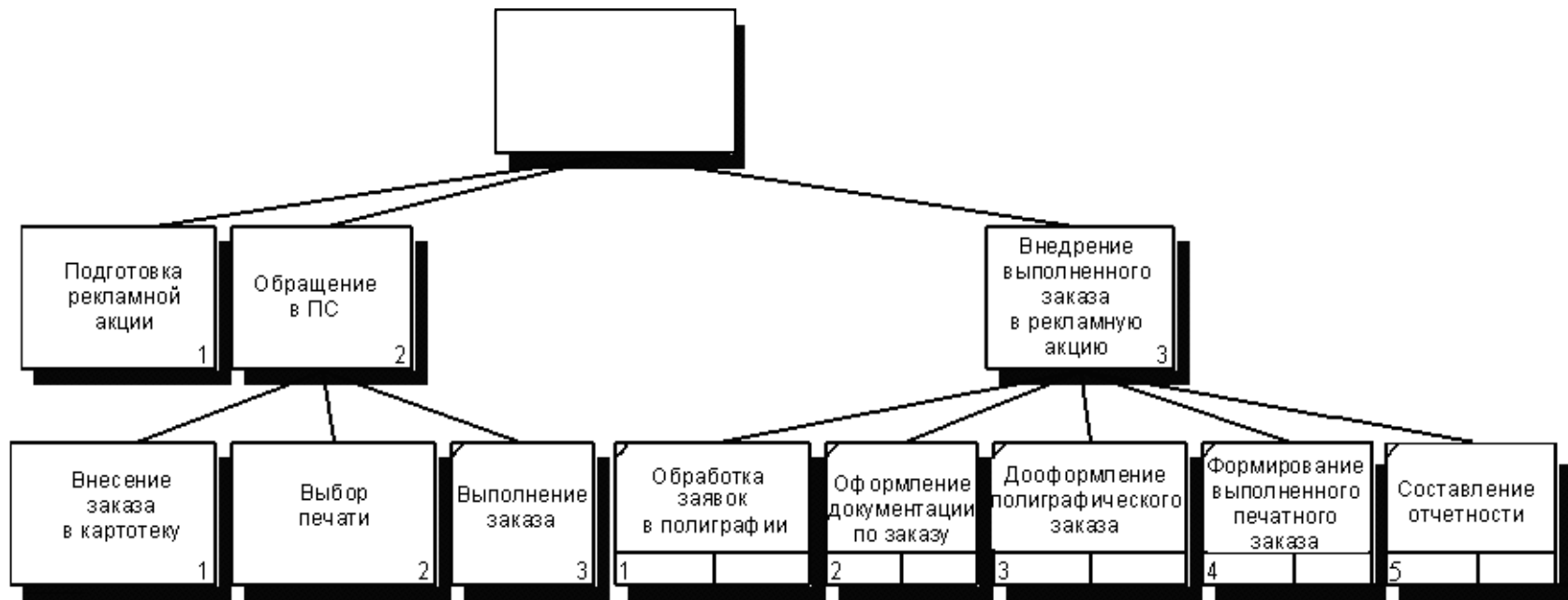


Рис. 4.2. Диаграмма дерева узлов к модели AS – IS

4.3. Редактор VISIO и нотация IDEF3

В VISIO отсутствует трафарет для построения диаграмм IDEF3, но это не значит что в этом редакторе невозможно реализовать их построение. При необходимости пользователь может построить шаблоны всех элементов, используемых при этом, и создать новый трафарет, наполненный этими элементами.

Вопросы для самоконтроля

1. Для чего предназначена диаграмма IDEF3?
2. Назовите основные элементы диаграммы IDEF3.
3. Что показывают связи в диаграммах IDEF3?
4. Перечислите типы стрелок в диаграммах IDEF3.
5. Для чего предназначен перекресток?
6. Назовите типы перекрестков, применяемых на диаграммах IDEF3.
7. Какой инструментарий используется для построения диаграмм IDEF3?

5. ОЦЕНИВАНИЕ ПРОЦЕССОВ МОДЕЛИ AS – IS В ALLFUSION PROCESS MODELER И ПОСТРОЕНИЕ МОДЕЛИ TO – BE

5.1. Оценивание бизнес-процессов в AllFusion Process Modeler

AllFusion Process Modeler представляет аналитику двух инструментов для оценивания модели – ABC (Activity Based Costing) и UDP (User Defined Properties) [4–6].

ABC может проводиться только тогда, когда создание модели закончено. ABC включает следующие основные понятия:

- объект затрат (например, готовое изделие);
- двигатель затрат (сырье, чертеж);
- центры затрат (статьи расхода).

Операции, используемые при стоимостном анализе, и последовательность их выполнения приведены в п. 1 «Стоимостный анализ» табл. 5.1 [2–4].

Таблица 5.1

Операции к стоимостному анализу

| Операция | Рекомендуемая последовательность действий. Примечания |
|---|---|
| 1. ABC (стоимостный анализ) | ABC (Activity Bases Costing) может проводиться только тогда, когда создание модели закончено |
| 1.1. Открытие модели и начальные установки | Меню: File → Open = <Модель, подлежащая оценке> → Model → Model Properties... → Model Properties → Display → [ABS data] → [Cost] → ABS Units – установить единицы измерения денег и времени |
| 1.2. Описание центров затрат | Меню: Model → Cost Center Editor → 1{Cost Center = <Имя статьи расхода> → Definition → <Описание статьи расхода> → [Add]}* – повторить для всех статей расхода → [Close] |
| 1.3. Задание стоимости работ (с самого нижнего уровня декомпозиции) | <Выбранная работа> → RClick → Costs = <В окне диалога внести расходы по статьям (Cost Centers), частоту проведения работы (Frequency) и продолжительность (Duration)> |
| 1.4. Задание режима подсчета затрат по работе | 1{[Compute from Decompositions] – автоматический [Over ride Decompositions] – вручную}* |

| Операция | Рекомендуемая последовательность действий. Примечания |
|---|---|
| 1.5. Получение отчета по стоимостному анализу | Меню: Tools → Reports ► → Activity Cost Report |
| 2. Внесение метрик-свойств | Если стоимостных показателей недостаточно, имеется возможность внесения собственных метрик-свойств (UDP). Для описания UDP служит диалог, выдаваемый командой меню: Model → UDP Definition Editor. После описания UDP каждой работе можно поставить в соответствие их набор. Для этого необходимо выполнить: <Выбранная работа> → DbClick → <Закладка UDP Values> = <Необходимые свойства>. Результат задания UDP можно проанализировать в отчете (<Меню> → Tools → Reports ► → Diagram Object Report) |

Если стоимостных показателей недостаточно, имеется возможность внесения собственных метрик-свойств, определенных пользователем (UDP). UDP позволяет провести дополнительный анализ, но без суммирующих подсчетов.

Общие указания по описанию UDP даны в п. 2 «Внесение метрик-свойств» табл. 5.1.

Имеется возможность задания 18 различных типов UDP (табл. 5.2), в том числе управляющих команд и массивов.

Таблица 5.2

Общие указания по описанию UDP

| Тип | Использование |
|----------------------------------|---|
| Text | При задании свойства стрелки или работы просто вносится текст. Например, это может быть просто дополнительное пояснение |
| Paragraph Text | Значение свойства этого типа – текст в несколько строк |
| Integer | Значение свойства этого типа – целое число, например, значение свойства «Количество баллов» |
| Command/ <i>Командная строка</i> | При задании значения UDP в списке свойств справа от имени свойства появляется кнопка >. При щелчке по этой кнопке выполняется командная строка. С помощью этого свойства можно связать с объектом модели документацию, хранящуюся в |

| | |
|--|---|
| | формате приложения Windows, например, Word, Excel и т. д. |
|--|---|

Окончание табл. 5.2

| Тип | Использование |
|--|--|
| Character | Значение свойства этого типа – один символ |
| Date mm/dd/yy (yy) | Значение свойства этого типа – дата |
| Real Number | Значение свойства этого типа – действительное число, например, значение свойства «Потребление электроэнергии, кВт/ч» |
| Text List (Single selection)/ <i>Массив строк</i> | Значения свойства этого типа должны быть определены в диалоге UDP Dictionary (поле Value). Объекту модели можно присваивать только одно значение из предварительно заданного списка |
| Integer List (Single selection)/ <i>Массив целых чисел</i> | |
| Command List/ <i>Массив команд</i> | |
| Date Listmm/dd/yy (yy) (Single selection)/ <i>Массив дат</i> | |
| Real Number List (Single selection)/ <i>Массив действительных чисел</i> | |
| Character List (Single selection)/ <i>Массив символов</i> | |
| Text List (Multiple elections)/ <i>Массив строк (множественный выбор)</i> | Значения свойства этого типа должны быть определены в диалоге UDP Dictionary (поле Value). Объекту модели можно присваивать одновременно несколько значений из предварительно заданного списка |
| Integer List (Multiple selections)/ <i>Массив целых чисел (множественный выбор)</i> | |
| Date List (Multiple selections)/ <i>Массив дат (множественный выбор)</i> | |
| Real Number List (Multiple selections)/ <i>Массив действительных чисел (множественный выбор)</i> | |

| | |
|--|--|
| Character List (Multiple selections)/Массив символов (множественный выбор) | |
|--|--|

Более полную информацию по описанию UDP можно найти, воспользовавшись обучающей программой к AllFusion Process Modeler либо литературой [4].

Для проведения более тонкого анализа можно применять специализированное средство стоимостного анализа EasyABC. AllFusion Process Modeler имеет двухнаправленный интерфейс с EasyABC. Для экспорта данных в EasyABC следует выбрать Меню: File → Export → Node Tree, а для импорта результирующих данных из EasyABC в AllFusion Process Modeler – Меню: File → Import → Cost.

Пример стоимостного анализа к фрагменту модели AS – IS в продолжение рассматриваемого примера. «Расщепляют» ранее созданную модель AS – IS, «отщипнув» от нее ветвь, начав с процесса «Выбор печати», используемого в качестве «корневого». При этом «корень» полученной «ветви» представлен одноименным контекстным процессом, показанным на рис. 5.1, с его декомпозицией на рис. 5.2.



Рис. 5.1. Контекстный процесс «Внедрение выполненного заказа в рекламную акцию»

Как пример ABC (затратного) анализа вычисляют затраты по процессу «Внедрение выполненного заказа в рекламную акцию».

В качестве исходных данных для анализа используют следующие сведения:

- оплата сотрудника (20 у.е./день);
- расходные материалы (150 у.е./заказ);

– амортизационные отчисления (10 у.е./заказ).

Затем выполняют следующие операции.

1. В закладке ABC Units диалога Model Properties устанавливают единицы измерения денег и времени.

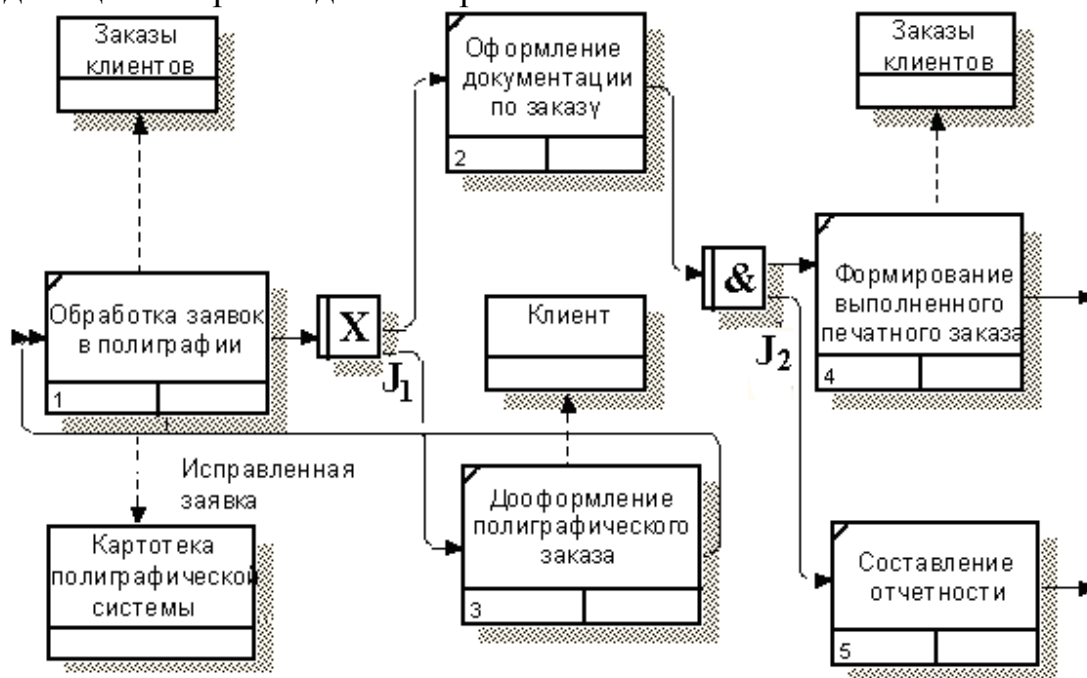


Рис. 5.2. Декомпозиция процесса «Внедрение выполненного заказа в рекламную акцию»

2. В диалоге ABC Cost Centers вносят название и определение центров затрат (табл. 5.3).

Таблица 5.3

Определение центров затрат

| Центр затрат | Определение |
|--------------|----------------------------|
| Зарплата | Оплата сотрудника |
| Материалы | Расходные материалы |
| Амортизация | Амортизационные отчисления |

3. Для отображения частоты или продолжительности переходят на закладку Display диалога Model Properties и переключают радиокнопки в группе ABC Units.

4. Для указания стоимости работы щелкают по ней правой кнопкой мыши и выбирают в контекстном меню Cost Editor.

Воспользовавшись этим приемом, вносят следующие параметры ABC (табл. 5.4).

Просматривают результат – стоимость работы верхнего уровня «Внедрение выполненного заказа в рекламную акцию».

На рис. 5.3, 5.4 показаны декомпозиция контекстной работы и сам процесс с указанием на значках затрат.

Таблица 5.4

Параметры к ABC-анализу

| Функциональный блок | Cost Centers | Затраты | Продолжительность | Частота |
|--|---|----------|-------------------|---------|
| Обработка заявок в полиграфии | Поскольку эта работа включает в себя выполнение одной из двух следующих работ, будем полагать, что частота выполнения каждой из них равна 0,5 | | | |
| Оформление документации по заказу | Зарплата | 20 у.е. | 1 день | 0,5 |
| Дооформление полиграфического за-каза | Зарплата | 20 у.е. | 1 день | 0,5 |
| Формирование выполненного печатного заказа | Зарплата | 800 у.е. | 1 заказ | 1,0 |
| | Материалы | 150 у.е. | 1 заказ | 1,0 |
| | Амортизация | 10 у.е. | 1 заказ | 1,0 |
| Составление отчетности | Зарплата | 20 у.е. | 1 заказ | 1,0 |

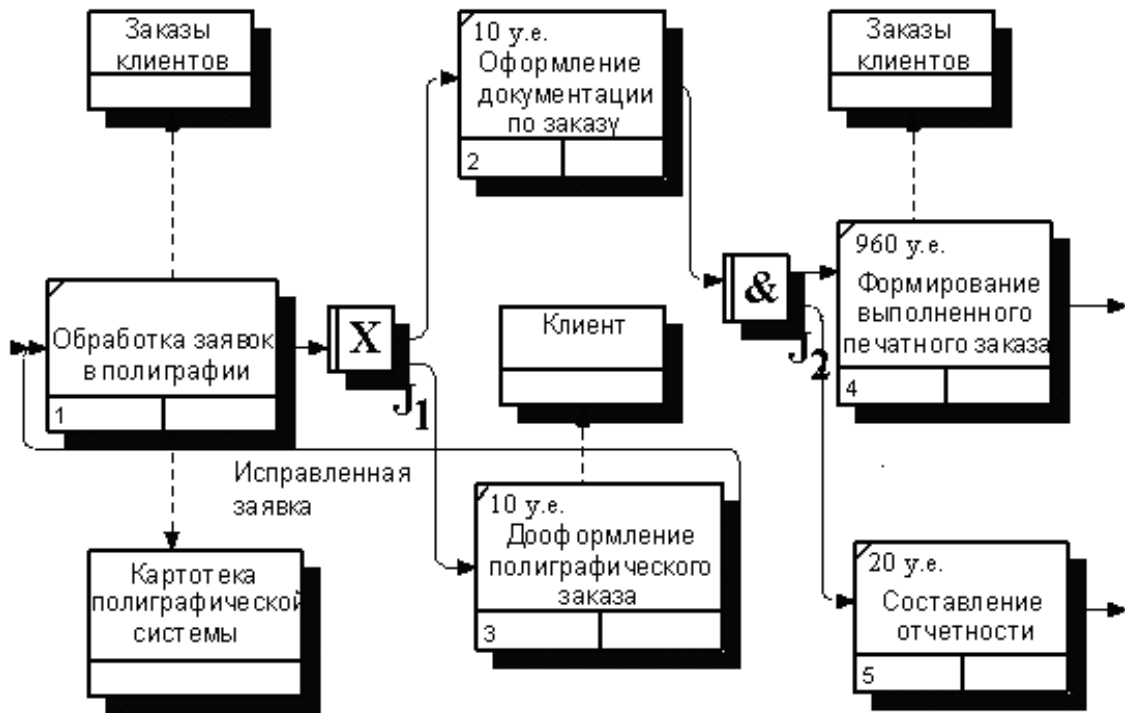


Рис. 5.3. Декомпозиция контекстного процесса «Внедрение выполненного заказа в рекламную акцию» после выполнения Cost-анализа

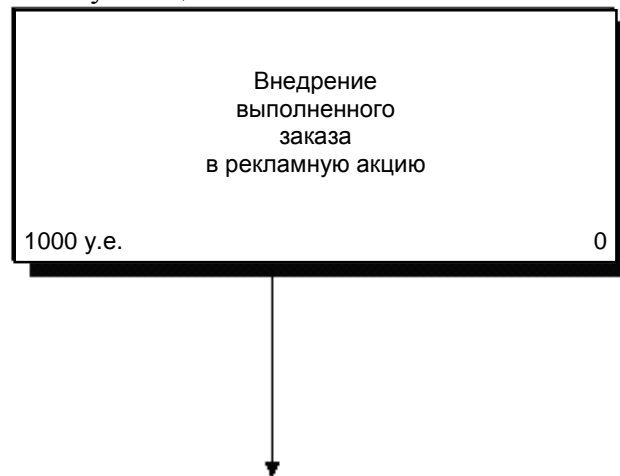


Рис. 5.4. Контекстный процесс «Внедрение выполненного заказа в рекламную акцию» после выполнения Cost-анализа

5.2. Построение модели ТО – ВЕ при разработке программного обеспечения «Информационная система обработки полиграфических заказов»

При построении модели ТО – ВЕ используют результаты обследования деятельности организации, точнее реинжиниринг созданной ранее модели AS – IS.

Начинают с создания контекстной диаграммы (рис. 5.5), на которой показано обслуживание клиента в полиграфической системе (ПС). В ПС поступают заказы от клиентов. Администратор принимает заказы и формирует из них очередь. После обработки ПС сформированных заказов формируется откорректированная информация о заказах и просходит внедрение полученных результатов заказов. Исполнителями процесса «Обслуживание клиента в полиграфической системе» являются три потока: «Администратор ПС», «Оператор ПС», «Служащий ПС».

Далее осуществляют декомпозицию процесса «Обслуживание клиента в полиграфической системе» на три процесса: «Подготовка рекламной акции», «Обращение в ПС», «Внедрение выполненного заказа в рекламную акцию» (рис. 5.6). Исполнителем процесса «Подготовка рекламной акции» является администратор ПС. В результате выполнения процесса на выходе получают информацию о заказе клиента и порядковый номер клиента, которые затем передаются в качестве управляющего процесса «Обращение в ПС».



Рис. 5.5. Контекстная диаграмма процесса «Обслуживание клиента в полиграфической системе»

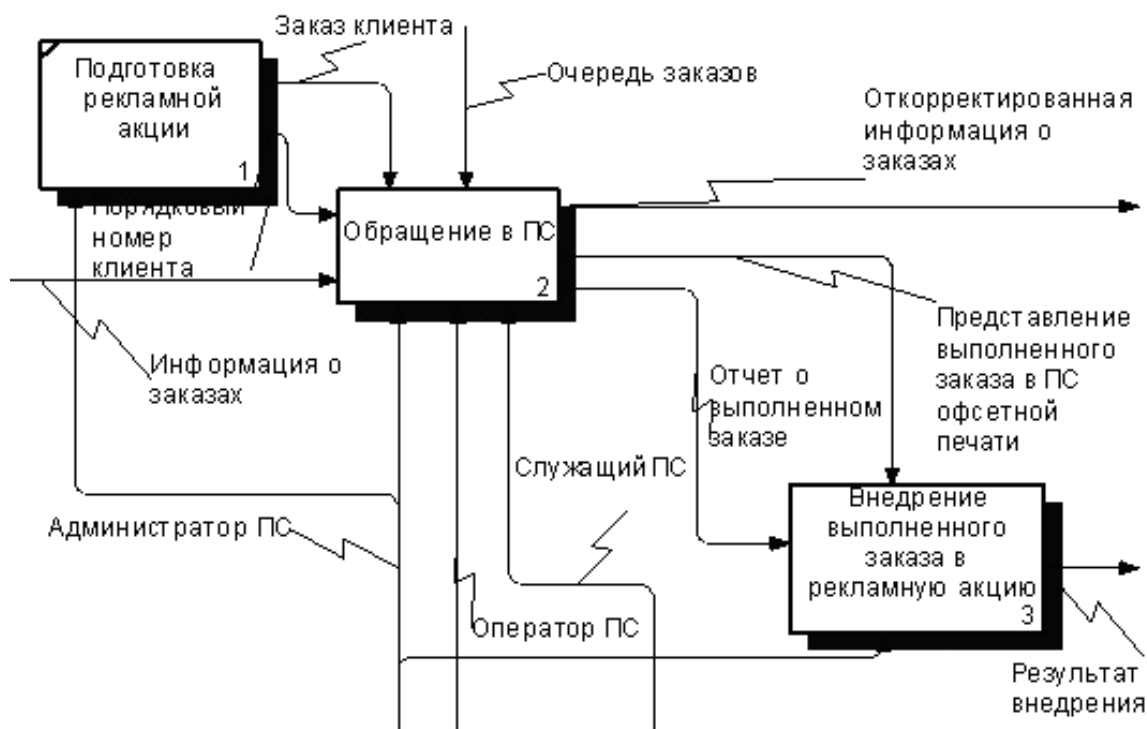


Рис. 5.6. Декомпозиция процесса «Обслуживание клиента в полиграфической системе»

В ходе реализации процесса «Обращение в ПС», исполнителем которого является администратор ПС, получают на выходе три потока: «Откорректированная информация о заказах», «Отчет о выполненном заказе» и «Представление выполненного заказа в ПС офсетной печати». Последних два потока передаются в качестве управляющих в процесс «Внедрение выполненного заказа в рекламную акцию».

На вход рассматриваемого процесса поступает информация о заказах. Управляют этим процессом два потока: «Очередь заказов» и «Заказ клиента».

В результате выполнения процесса «Внедрение выполненного заказа в рекламную акцию», исполнителем которого является администратор, получают поток «Результат внедрения». Управляет этим процессом поток «Представление выполненного заказа в ПС офсетной печати».

Затем переходят к декомпозиции процесса «Обращение в полиграфическую систему» на три процесса: «Внесение заказа в БД», «Выбор печати», «Выполнение заказа» (рис. 5.7). Управляет процессом «Внесение заказа в БД» два потока: «Очередь заказов», «Заказ клиента».

На вход указанного процесса подается «Информация о заказах» и «Порядковый номер клиента». В результате исполнителем процесса «Внесение заказа в БД» является администратор ПС.

В ходе выполнения процесса на выходе получают два потока: «Откорректированная информация о заказах» и «Внесенные заказы». Последний в свою очередь управляет процессом «Выбор печати».

Выполнение процесса «Выбор печати», исполнителями которого являются администратор ПС и оператор ПС, дает на выходе два потока: «Представление выполненного заказа в ПС офсетной печати», который затем переходит к управлению процессом «Внедрение выполненного заказа в рекламную акцию», и «Выбранный тип печати». Поток «Выбранный тип печати» является управляющим процесса «Выполнение заказа». Исполнителями процесса «Выполнение заказа» является администратор ПС и служащий ПС.

При осуществлении декомпозиции процесса «Внесение заказа в БД» DF-диаграмма разбивается на три процесса: «Внести заказ в глобальную БД ПС», «Принять заказ или отклонить», «Сортировка в БД заказов» (рис. 5.8).



Рис. 5.7. Декомпозиция процесса «Обращение в полиграфическую систему»

В ходе выполнения процесса «Внести заказ в глобальную БД ПС» заказ вносится в основную картотеку ПС. Входными потоками процесса являются «Заказ клиента» и «Очередь заказов». Последний извлекается из «Неотсортированной БД», представленной на диаграмме в виде хранилища 1. Также подается поток «Порядковый номер клиента».

Выходом из процесса является поток «Откорректированная информация о заказах», которая подается на вход процесса «Принять заказ или отклонить».

После выполнения процесса «Принять заказ или отклонить» получают на выходе два потока: «Откорректированная информация о заказах» и «Оформленные заказы». Последний подается на процесс «Сортировка в БД заказов».

В результате выполнения процесса «Сортировка в БД заказов» получают два потока: «Внесенные заказы» и «Сформированная очередь заказов». Последний помещается в отсортированную БД. На вход рассматриваемого процесса, помимо потока «Оформленные заказы», подается поток «Информация о заказах».

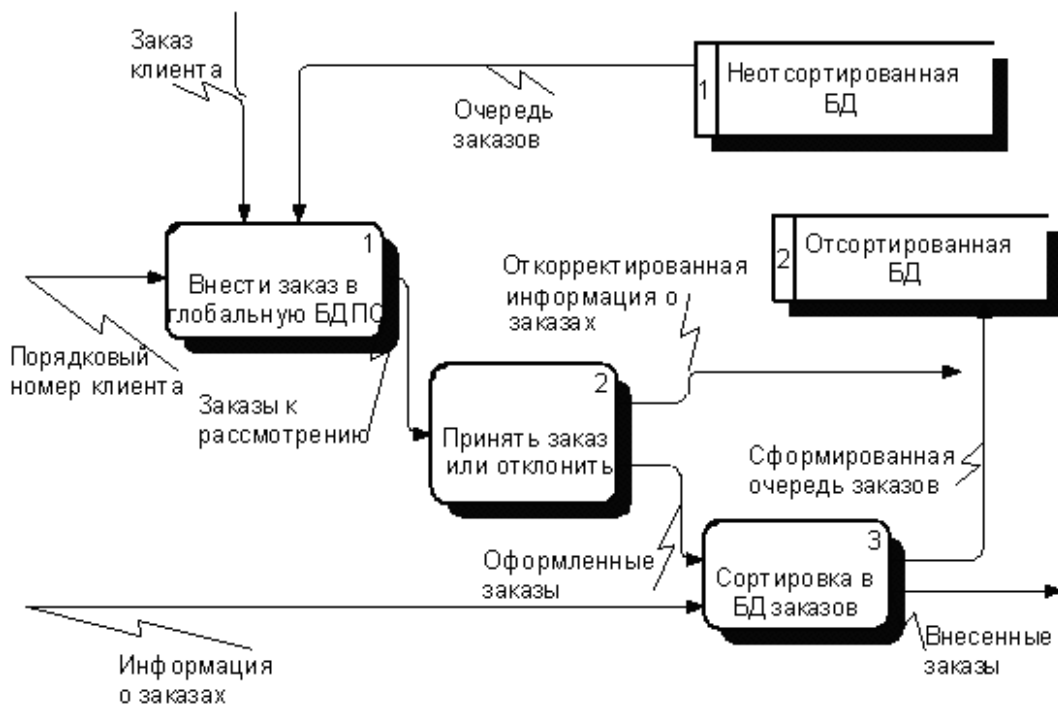


Рис. 5.8. Декомпозиция процесса «Внесение заказа в базу данных» в DF-диаграмму

При осуществлении декомпозиции процесса «Выбор печати» DF-диаграмма разбивается на три процесса: «Выбор типа и формы печати заказа», «Выбор печатного оборудования», «Выполнение заказа офсетной печатью» (рис. 5.9).

Процесс «Выбор типа и формы печати заказа» вносит заказ в базу данных клиента. Входными потоками процесса являются «Внесенные заказы» и «Принятые заказы».

Последние извлекаются из «БД заказов клиентов». Выходом из рассматриваемого процесса является поток «Выбор печатного оборудования».

В ходе выполнения процесса «Выбор печатного оборудования» получают два выходных потока: «Выполненные заказы» и «Заказ в печать». Последний является входом процесса «Выполнение заказа офсетной печатью».

В результате выполнения процесса «Выполнение заказа офсетной печатью» по входному потоку «Заказ в печать» получают два потока: «Представление выполненного заказа в ПС офсетной печати» и «Выбранный тип печати». Последний помещается в базу данных к рекламной акции.

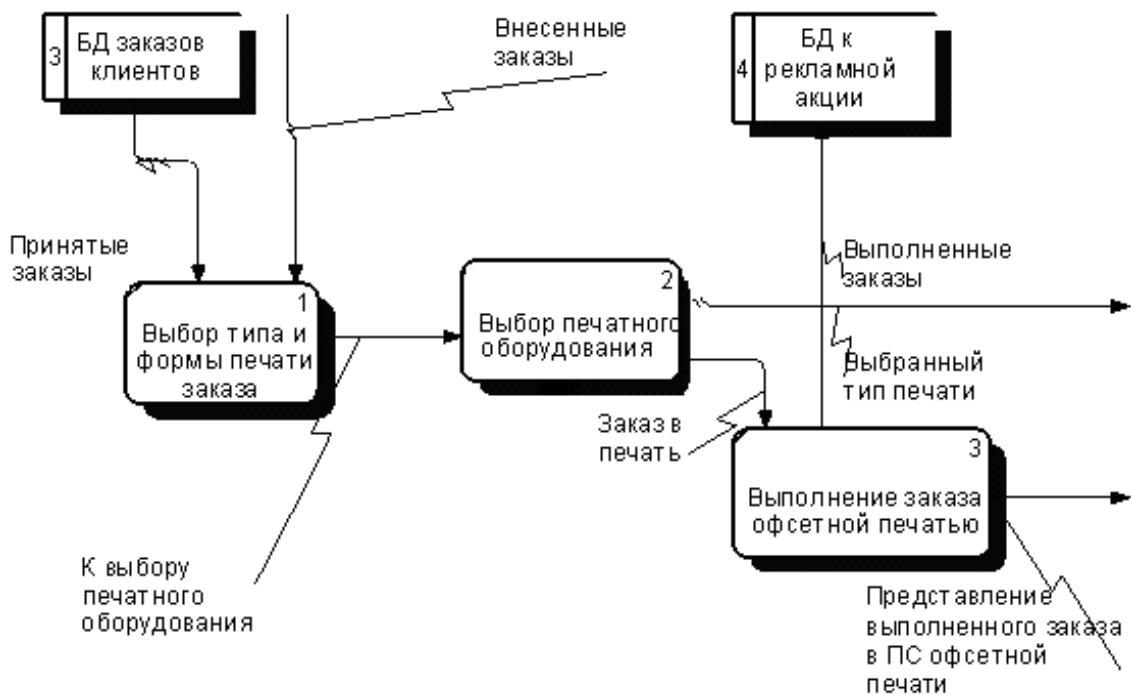


Рис. 5.9. Декомпозиция процесса «Выбор печати» в DF-диаграмму

При декомпозиции процесса «Выполнение заказа» IDEF3-диаграмма включает в себя пять работ: «Обработка заявок в полиграфии», «Оформление документации по заказу», «Дооформление полиграфического заказа», «Формирование выполненного печатного заказа» и «Составление отчетности» (рис. 5.10).

Видно, что работа «Обработка заявок в полиграфии» требует выполнения одной из двух работ (либо «Оформление документации по заказу», либо «Дооформление полиграфического заказа»).

Работа «Оформление документации по заказу» передает управление на две параллельные работы: «Формирование выполненного печатного заказа» и «Составление отчетности».

Причем работа «Обработка заявок в полиграфии» обращается к двум ссылочным объектам: «Заказы клиентов» и «База данных полиграфической системы». А работа «Дооформление полиграфического заказа» – к ссылочному объекту «Клиент». Работа «Формирование выполненного печатного заказа» – к ссылочному объекту «Заказы клиентов».

Перекресток J_1 позволяет реализовать работу «Оформление документации по заказу» либо работу «Дооформление полиграфического заказа».

Работа «Оформление документации по заказу» передает управление на две параллельные работы: «Формирование выполненного печатного заказа» и «Составление отчетности».

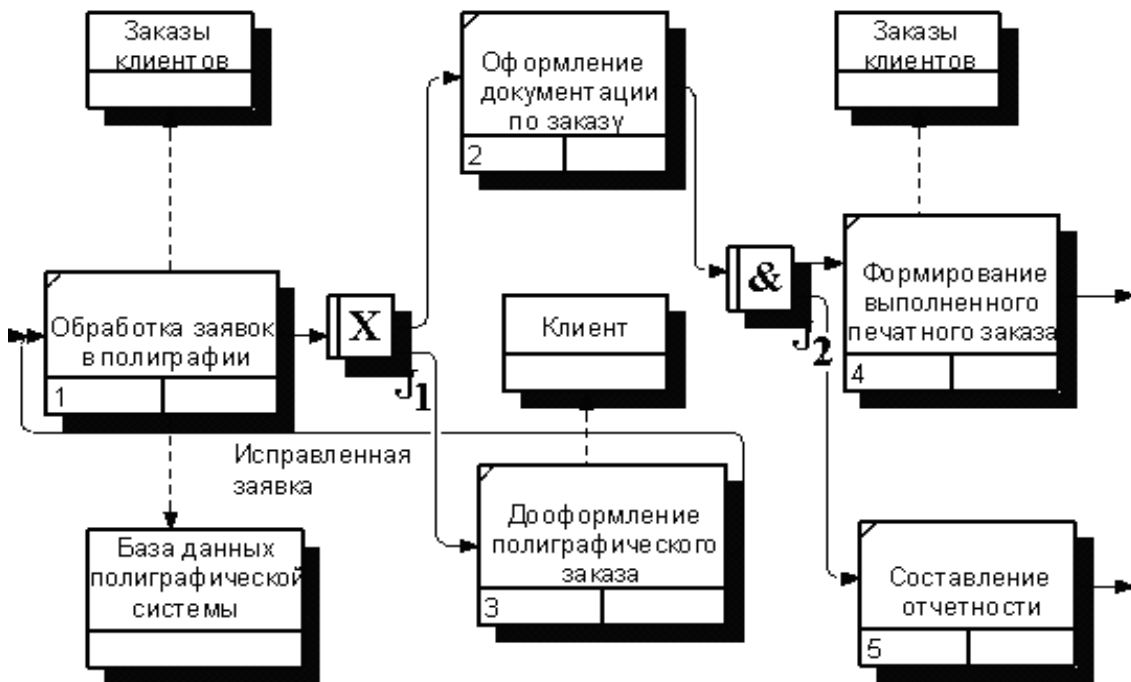


Рис. 5.10. Декомпозиция процесса «Выполнение заказа» в IDEF3-диаграмму

В завершение создания модели ТО – ВЕ строят диаграмму дерева узлов, которая может быть представлена в виде, показанном на рис. 5.11.

На основании модели формулируют цели и задачи предмета разработки.

Целью разработки программного обеспечения «Информационная система обработки полиграфических заказов» является повышение обработки и хранения данных о заказах.

В соответствии с поставленной целью разрабатываемое ПО должно решать следующие задачи:

- 1) ввести новый заказ на типоофсетном печатном аппарате;
- 2) изменить существующий заказ;
- 3) выполнить заказ офсетной печатью;
- 4) обновить базы данных типографии;
- 5) оформить заказ на глубокую печать;
- 6) отклонить заказ на рулонных машинах.

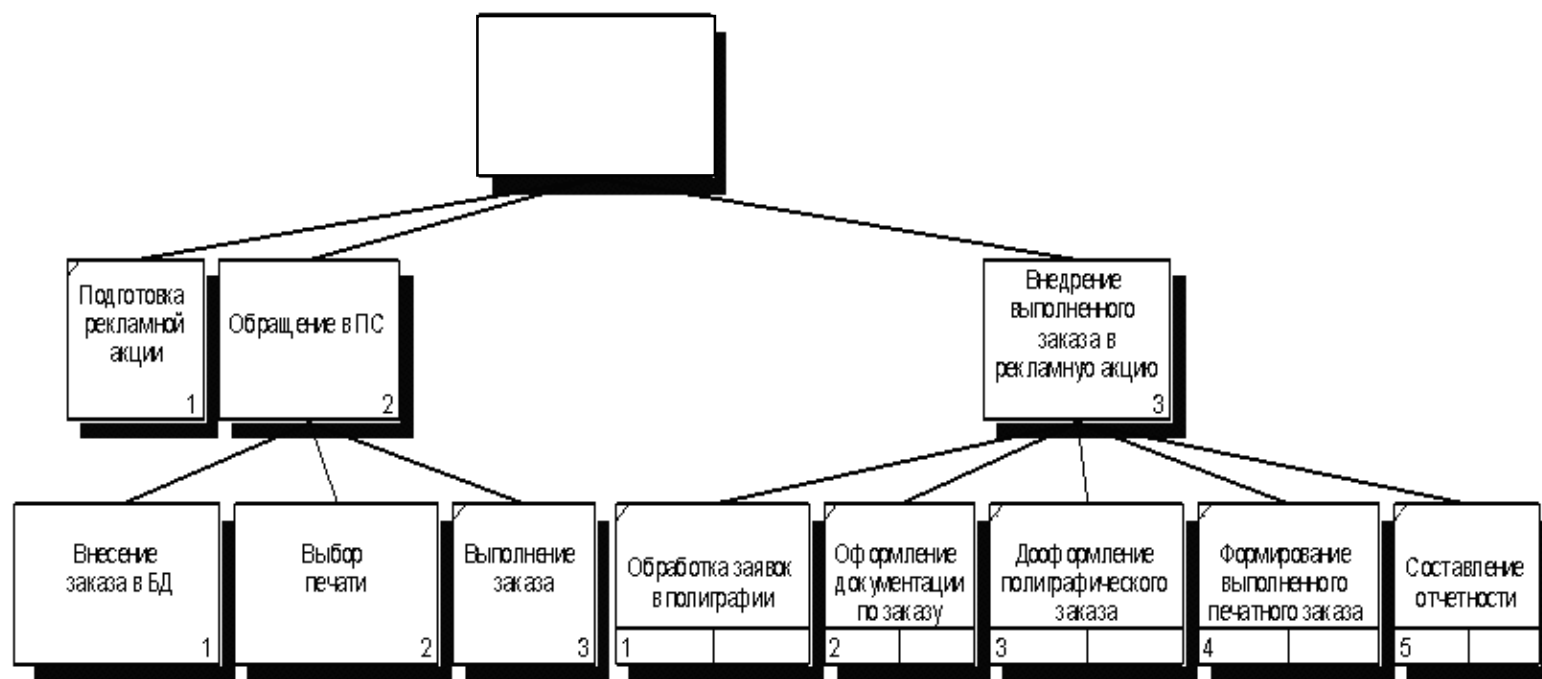


Рис. 5.11. Диаграмма дерева узлов к модели ТО – ВЕ

Вопросы для самоконтроля

1. Перечислите и охарактеризуйте инструменты оценивания бизнес-процессов, представляемые AllFusion Process Modeler.
2. Какое место занимает в моделях AS – IS и TO – BE оценивание бизнес-процессов?
3. Что позволяет показать модель AS – IS?
4. Чем отличается модель TO – BE от AS – IS и в чем заключается ее роль в процессе разработки ПО?
5. Какой модели предшествует обследование деятельности, а какой – ее критический анализ?
6. Какие из рассмотренных – IDEF0, DFD, IDEF3 – диаграммных техник используются при построении моделей AS – IS и TO – BE?
7. На основании какой модели формируют цели и задачи предмета разработки?

6. МОДЕЛИРОВАНИЕ ДАННЫХ

6.1. Построение ER-модели данных

Для разработки моделей данных предназначены диаграммы «сущность – связь». Они обеспечивают стандартный способ определения данных и отношений между ними. С помощью ERD осуществляется детализация хранилищ данных проектируемой системы, а также документируются сущности системы и способы их взаимодействия, включая идентификацию объектов, важных для предметной области (сущностей), свойств этих объектов (атрибутов) и их отношений с другими объектами (связей) [1, 2].

Как следует из названия диаграммы, основными элементами ERD являются сущность и связь.

Сущность представляет собой множество экземпляров реальных или абстрактных объектов, обладающих общими атрибутами или свойствами.

Связь идентифицирует требования, в соответствии с которыми сущности вовлекаются в отношения, т. е. сущности представляют собой базовые типы информации, хранимой в базе данных, а отношения показывают, как эти типы взаимодействуют друг с другом.

Что касается типа сущностей, то они подразделяются на независимые, зависимые и ассоциативные.

Независимая сущность представляет независимые данные, которые всегда присутствуют в системе. При этом отношения с другими сущностями могут как существовать, так и отсутствовать.

Зависимая сущность – это данные, зависимые от других сущностей в системе. Они всегда имеют отношения с другими сущностями.

Ассоциативная сущность представляет данные, которые ассоциируются с отношениями между двумя и более сущностями.

Как видим, тип сущности определяется ее связью с другими сущностями.

Идентифицирующая связь устанавливается между независимой (родительской) и зависимой (дочерней) сущностями.

Неидентифицирующая связь служит для связывания независимых сущностей.

Значение (мощность) связи характеризуется ее типом и, как правило, выбирается из следующего множества:

{«0 или 1», «0 или более», «1», «1 или более», «р:q» (диапазон)}.

Пара значений связей, принадлежащих одному и тому же отношению, определяет тип этого отношения. Для большинства приложений достаточно использовать следующие типы отношений:

1) 1×1 (один к одному). Отношения данного типа применяются, как правило, на верхних уровнях иерархии модели данных;

2) $1 \times n$ (один ко многим). Отношения этого типа являются наиболее часто используемыми;

3) $n \times m$ (многие ко многим). Отношения данного типа обычно применяются на ранних этапах проектирования (при построении логической модели данных).

ER-диаграмма может быть представлена на уровне сущностей, атрибутов, первичных ключей и т. д. Уровень атрибутов позволяет детализировать сущности.

Что касается нотаций, используемых при построении ER-диаграмм, то это нотация Чена, Барнера, IDEF1X, IE, DM. Наибольший интерес представляют три последние международно-признанные и используемые в AllFusion ERwin Data Modeler нотации:

– *Integration DEfinition for Information Modeling (IDEF1X)*. Эта нотация была разработана по заказу вооруженных сил США и используется не только ими, но и НАТО, Международным валютным фондом и др.;

– *Information Engeneering (IE)*. Нотация была разработана Мартином Финкельштейном и другими авторами и используется в основном в промышленности;

– *Dimensional Modeling (DM)*. Эта специализированная нотация предназначена для разработки хранилищ данных.

Графическое изображение элементов ERD (на примере IDEF1X) будет показано далее.

6.2. Этапы разработки диаграммы «сущность – связь»

Разработка ERD включает следующие основные этапы.

1. Идентификация сущностей, их атрибутов, а также первичного и альтернативного ключей.

2. Идентификация отношений между сущностями и указание типов отношений.

3. Разрешение неспецифических отношений (отношений $n \times m$).

Этап 1. Данный этап является определяющим при построении модели. Его исходной информацией служит содержимое хранилищ данных, определяемое входящими и выходящими в/из него потоками данных. Ниже на рис. 6.1 приведен фрагмент DF-диаграммы, моделирующей деятельность бухгалтера предприятия.



Рис. 6.1. Фрагмент DF-диаграммы, моделирующей деятельность бухгалтера

Его единственное хранилище «Данные о персонале» должно содержать информацию о всех сотрудниках: их имена, адреса, должности, оклады и т. п.

Первоначально осуществляется анализ хранилища, включающий сравнение содержимого входных и выходных потоков и создание на основе этого сравнения схемы хранилища. Перечислим структуры данных, содержащиеся во входных и выходных потоках.

Входные структуры данных

вновь_принятые
дата_найма
фамилия
таб_номер

Выходные структуры данных

адрес_служащего
фамилия
адрес
подробности з/пл

| | |
|------------------|-------------------|
| адрес | фамилия |
| должность | таб_номер |
| начальная_з/пл | текущая з/пл |
| уволенные | история занятости |
| фамилия | фамилия |
| таб_номер | таб_номер |
| изменение_адреса | дата_найма |
| фамилия | история_карьеры* |
| таб_номер | должность |
| старый_адрес | дата_изменения |
| новый_адрес | история з/пл |
| изменение_з/пл | з/пл |
| фамилия | |
| таб_номер | |
| старая_з/пл | |
| новая_з/пл | |
| дата_изменения | |

Сравнивая входные и выходные структуры, отметим следующие моменты.

1. Поле **адрес** хранит текущий адрес сотрудника, а структура **изменение_адреса** хранит и старый адрес, что не является необходимым, исходя из выходных потоков.

2. **История_з/пл**, наоборот, требует перечень всех окладов сотрудника, поэтому необходимо иметь набор, состоящий из пар (**з/пл**, **дата**), а не просто **старая_з/пл** и **новая_з/пл** (как во входном потоке).

3. Аналогичная ситуация и с **историей_карьеры***. Отметим, что на диаграмме вообще отсутствует поток, определяющий изменение в должности, т. е. обнаружено серьезное упущение в функциональной модели.

4. Подчеркнем, что изменение в должности обычно (но не всегда) соответствует изменению зарплаты.

С учетом этих моментов первый вариант схемы может выглядеть следующим образом:

фамилия
таб_номер
адрес
текущая_з/пл
дата_найма

история_карьеры*

должность

дата_изменения

история_з/пл

з/пл

дата_изменения

На следующем шаге упрощается схема за счет устранения избыточности. Действительно, **текущая_з/пл** всегда является последней записью в **истории_карьеры***, а **дата_найма** содержится в разделе **история_з/пл** и **история_карьеры***. Кроме того, несколько дат в последних разделах одни и те же, поэтому целесообразно создать на их основе структуру **история_з/пл_карьеры** и вводить в нее данные при изменении **должности** и/или **зарплаты**:

фамилия

таб_номер

адрес

история_з/пл_карьеры*

з/пл

должность

дата_изменения

Следующий шаг – упрощение схемы при помощи нормализации (удаления повторяющихся групп).

Единственным способом нормализации является расщепление данной схемы на две более простые, показанные на рис. 6.2.

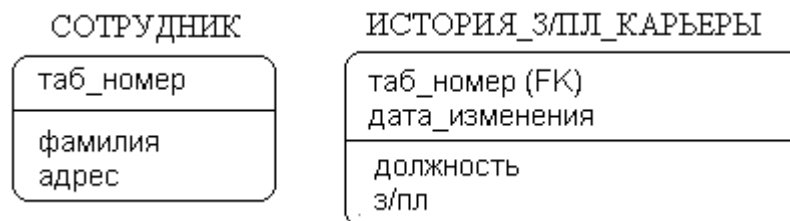


Рис. 6.2. Сущности модели без связи (уровень атрибутов)

Концепции и методы нормализации были разработаны Коддом, который установил существование трех типов нормализованных схем, названных в порядке уменьшения сложности первой, второй и третьей нормальной формой (соответственно 1НФ, 2НФ, 3НФ).

На практике отношения 1НФ и 2НФ имеют тенденцию возникать при попытке описать несколько реальных сущностей в одной схеме (заказ и книга, проект и сотрудник). 3НФ является

наиболее простым способом представления данных, отражающим здравый смысл.

Построив ЗНФ, мы фактически выделяем базовые сущности предметной области.

Этап 2. Этот этап служит для выявления и определения отношений между сущностями. На данном этапе некоторые отношения могут быть неспецифическими ($n \times m$ – многие ко многим). Такие отношения потребуют дальнейшей детализации на этапе 3.

Определение отношений включает выявление связей. Для этого отношение должно быть проверено в обоих направлениях следующим образом: выбирается экземпляр одной из сущностей и определяется, сколько различных элементов 2-й сущности может быть связано и наоборот. Рассмотрим отношение между сущностями «Сотрудник» и «История_з/пл_карьеры».

У отдельного сотрудника должность и/или зарплата может меняться ноль, один или много раз, порождая соответствующее число экземпляров сущности «История_з/пл_карьеры». С другой стороны, каждый экземпляр этой сущности соответствует только одному конкретному сотруднику. Поэтому между этими двумя сущностями имеется отношение типа $1 \times n$ (один ко многим) со связью «Один» на конце отношения у сущности «Сотрудник» и со связью «Ноль, один или много» на конце у сущности «История_з/пл_карьеры». Что и зафиксировано ниже на диаграмме уровня атрибутов (рис. 6.3).



Рис. 6.3. Модель данных на уровне атрибутов

Этап 3. Названный этап предназначен для разрешения неспецифических (многие ко многим) отношений. Для этого каждое неспецифическое отношение преобразуется в два специфических отношения с введением новых (а именно, ассоциативных) сущностей. Рассмотрим пример решения неспецифического отношения, показанного на рис. 6.4.



Рис. 6.4. Неспецифическое отношение

Неспецифическое отношение, приведенное выше, показывает, что студент может изучать много предметов, а предмет может изучаться многими студентами.

Однако мы не можем определить, какой студент изучает какой предмет, пока не введем для разрешения этого неспецифического отношения третью (ассоциативную) сущность «Изучение предмета» (рис. 6.5).

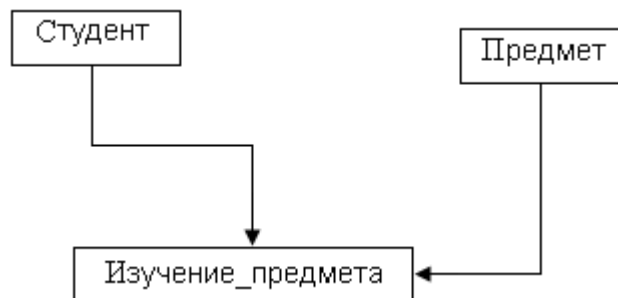


Рис. 6.5. Разрешение неспецифического отношения

Каждый элемент введенной сущности связан с одним студентом и с одним предметом. Таким образом, ассоциативные сущности по своей природе являются представлениями пар реальных ответов и обычно появляются на этапе 3.

Вопросы для самоконтроля

1. Перечислите основные этапы разработки ER-диаграммы.
2. Какая модель бизнес-процесса является отправной при построении модели данных?
3. Назовите, какой составляющей модели ТО – ВЕ среди диаграмм IDEF0, DFD, IDEF3 следует отдать предпочтение при моделировании данных?
4. Что из обследования деятельности предприятия может понадобиться при построении ER-диаграммы?

7. ЛОГИЧЕСКОЕ МОДЕЛИРОВАНИЕ ДАННЫХ

7.1. Инструментальная среда AllFusion ERwin Data Modeler

AllFusion ERwin Data Modeler является новым именем известного ERwin. Это Case-средство предназначено для моделирования модели – логической и физической. AllFusion ERwin Data Modeler позволяет проводить процессы прямого и обратного проектирования БД. Это означает, что по физической модели данных можно сгенерировать системный каталог базы данных или автоматически создать модель данных на основе информации системного каталога. Кроме того, AllFusion ERwin Data Modeler дает возможность выравнивать модель и содержимое системного каталога после редактирования того либо другого [1, 2, 4].

Ниже приведен список перемещаемых панелей инструментов, а также основные не зависящие от уровня моделирования данных операции.

Перемещаемые панели инструментов:

Standard – стандартная;

Toolbox – палитра инструментов;

Transforms – трансформация;

Drawing – рисование;

Aliqment – выравнивание;

Database – работа с сервером базы данных;

Model Mart – работа с хранилищем модели;

Fount & Color – выбор шрифтов и цвета.

Операции, не зависящие от уровня моделирования данных, и рекомендуемая последовательность их выполнения даны в табл. 7.1 [4, 19].

Таблица 7.1

Операции, предшествующие моделированию данных

| Операция | Рекомендуемая последовательность действий. Примечания |
|--|--|
| 1. Создание новой/открытие существующей модели | <ОД создание/открытие модели> → {[Create a new model] [Open on Existing file]} → [OK] |

| | |
|----------------------------|---|
| 2. Выбор типа новой модели | <ОД Create Model – Select Template> → {[Logical] [Logical/Physical]} → [OK] |
|----------------------------|---|

Окончание табл. 7.1

| Операция | Рекомендуемая последовательность действий. Примечания |
|---|---|
| 3. Отображение всех объектов на уровне модели | Model Explorer: <Вкл. Model> |
| 3.1. Выбор контекстного меню объекта | <Вкл. Model> → <Требуемый объект> → RClick |
| 3.2. Редактирование объекта | <Контекстное меню объекта> → Properties |
| 3.3. Манипулирование объектами | Атрибуты (колонки) можно переносить из сущности (таблицы) в сущность (таблицу) внутри Model Explorer методом drag & drag |
| 4. Отображение и редактирование подмножеств модели | Model Explorer: <Вкл. Subject Areas> – для копирования сущности из одного подмножества модели в другое. Можно воспользоваться методом drag & drag |
| 5. Отображение доменов | <Model Explorer> → <Вкл. Domains> |
| 6. Установка цвета и шрифта | {<Панель инструментов Font & Color> <Необходимый элемент> → RClick → Font/Color <Меню> → Format → Default Font & Colors <Свободное поле диаграммы> → RClick → Font/Color} |
| 7. Выравнивание объектов на диаграмме | <Выделяем выравниваемые объекты> → <Панель выравнивания Aliqment> |
| 8. Создание, удаление и редактирование подмножеств модели | Меню: Model → Subject Areas → Диалог Subject Areas |
| 9. Создание хранимого отображения | Меню: Format → Stored Display Settings → <Диалог Stored Displays> |
| 10. Вызов обучающей программы | Меню: Tutorial → Диалог ERwin Tutorial (18 уроков) |

7.2. Разработка логической модели данных в AllFusion ERwin Data Modeler

Рассматриваемое инструментальное средство позволяет представить модель данных как на физическом, так и на логическом

уровнях. На логическом уровне данные не связаны с конкретной системой управления базы данных (СУБД) и могут быть проиллюстрированы даже для неспециалистов.

При этом различают три основных уровня логической модели, отличающихся по глубине представления информации о данных:

- 1) диаграмма «сущность – связь» (Entity Relationship Diagram, ERD);
- 2) модель данных, основанная на ключах (Key Based model, KB);
- 3) полная атрибутивная модель (Fully Attributed model, FA).

Ниже приведен список кнопок палитры инструментов, а также операции, используемые при логическом моделировании данных, и последовательность их выполнения.

Кнопки палитры инструментов (TollBox):

- [Select] – установка фокуса на каком-либо объекте модели;
- [Entity] – создание новой сущности;
- [Complete sub-category] – создание категории;
- [Identifying relationship] – создание идентифицирующей связи;
- [Many to many] – создание связи «многие ко многим»;
- [Non identifying relationship] – создание неидентифицирующей связи.

Операции к логическому моделированию данных и последовательность их выполнения представлены в табл. 7.2.

Таблица 7.2

Операции к логическому моделированию данных

| Операция | Рекомендуемая последовательность действий. Примечания |
|--|--|
| 1. Общие установки | Тип модели – Logical |
| 1.1. Задание нотации | Меню: Model → Model Properties → <Вкл. Notations> = {[IDEF1X] [IE]} |
| 1.2. Переключение между уровнями отображения диаграммы | {Панель инструментов: {[Entity] [Attribute] [Definition]} <Свободное поле диаграммы> → RClick → Display Level → <Необходимый уровень отображения: Entity, Attribute, Primary Key, Definition, Icon>} |
| 2. Работа с сущностями | Тип модели – Logical; уровень – Entity |
| 2.1. Внесение сущности в модель | Палитра инструментов: [Entity] → <Необходимое место в окне диаграммы> → Click |
| 2.2. Задание свойств сущности | <Выбранная сущность> → RClick → Entity Properties → Диалог Entities – дать описание сущности, используя предложение закладки |
| 3. Работа со связями | |

| | |
|---------------------------|---|
| 3.1. Создание новой связи | Палитра инструментов: {[Identifying Relationship] [Non Identifying Relationship]} → Окно диаграммы: <Родительская сущность> → Click → <Дочерняя сущность> → Click |
|---------------------------|---|

Окончание табл. 7.2

| Операция | Рекомендуемая последовательность действий. Примечания |
|--|--|
| 3.2. Редактирование свойств связи | <Выбранная связь> → RClick → Relationship Properties → Диалог Relationships |
| 3.3. Отображение имени связи | <Свободное место диаграммы> → RClick → Relationship Display → Verb Phrase |
| 3.4. Отображение мощности связи | <Свободное место диаграммы> → RClick → Relationship Display → Cardinality |
| 4. Работа с атрибутами | Тип модели – Logical; уровень – Attribute |
| 4.1. Описание атрибутов | <Выбранная сущность> → RClick → Attributes → <Диалог Attributes > |
| 4.2. Перенос атрибута внутри и между сущностями | <Выбранный атрибут> → Click – указатель приобретает вид кисти руки, после чего можно воспользоваться методом drag & drag для переноса атрибута |
| 4.3. Отображение полного имени атрибута (имени и роли) | <Свободное место диаграммы> → RClick → Entity Display → Rolename/Attribute |
| 4.4. Задание режима именованния атрибутов | Меню: Names → Model Naming Options → <Диалог Model Naming Options> → <Вкл. Duplicate Names> |

7.3. Построение логической модели данных при разработке программного обеспечения «Информационная система обработки полиграфических заказов»

Проанализировав входные/выходные потоки и выполнив необходимые операции, связанные с устранением избыточности, выделяют следующие сущности: «Издат-полиграф», «Заказы», «Заказчики», «Выдача заказов».

После выявления отношений между сущностями, идентификации типов отношений и разрешения неспецифических отношений получают окончательную логическую модель данных,

которая может быть представлена в виде следующей ER-диаграммы на следующих уровнях:

- 1) на уровне сущностей (рис. 7.1);
- 2) на уровне атрибутов (рис. 7.2);
- 3) на уровне первичных ключей (рис. 7.3);
- 4) на уровне определений (рис. 7.4);
- 5) на уровне презентаций (рис. 7.5).

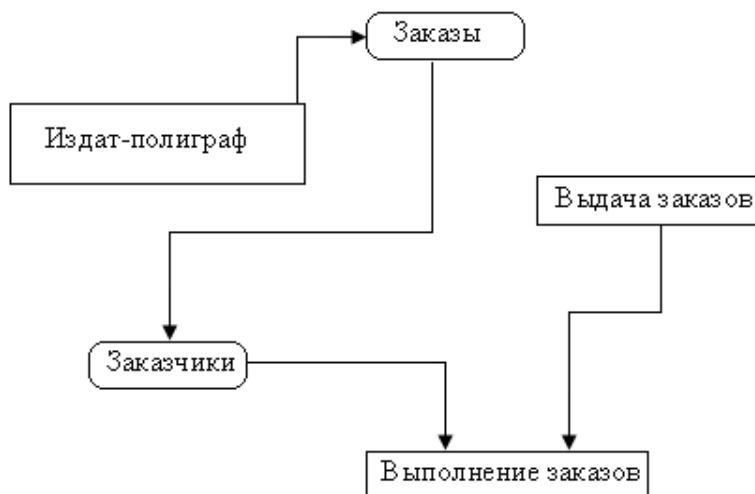


Рис. 7.1. Логическая модель данных на уровне сущностей

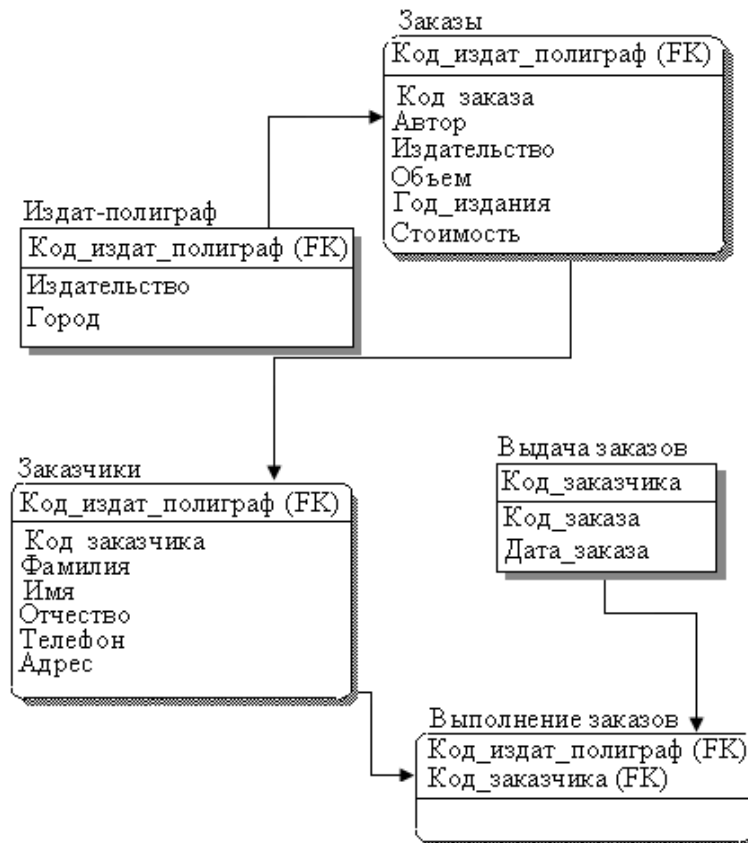


Рис. 7.2. Логическая модель данных на уровне атрибутов

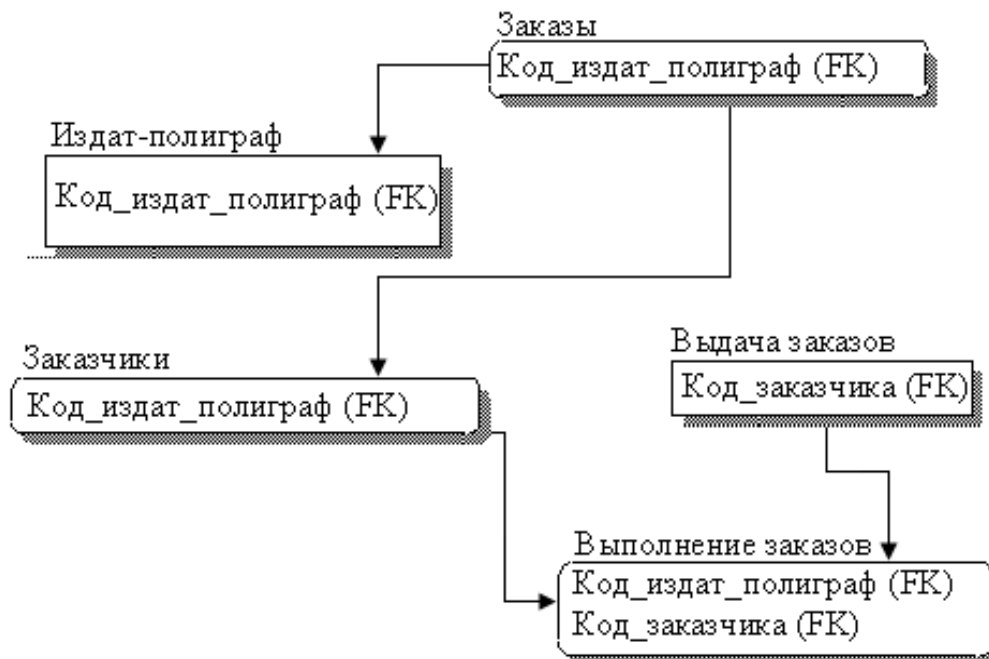


Рис. 7.3. Логическая модель данных на уровне первичных ключей

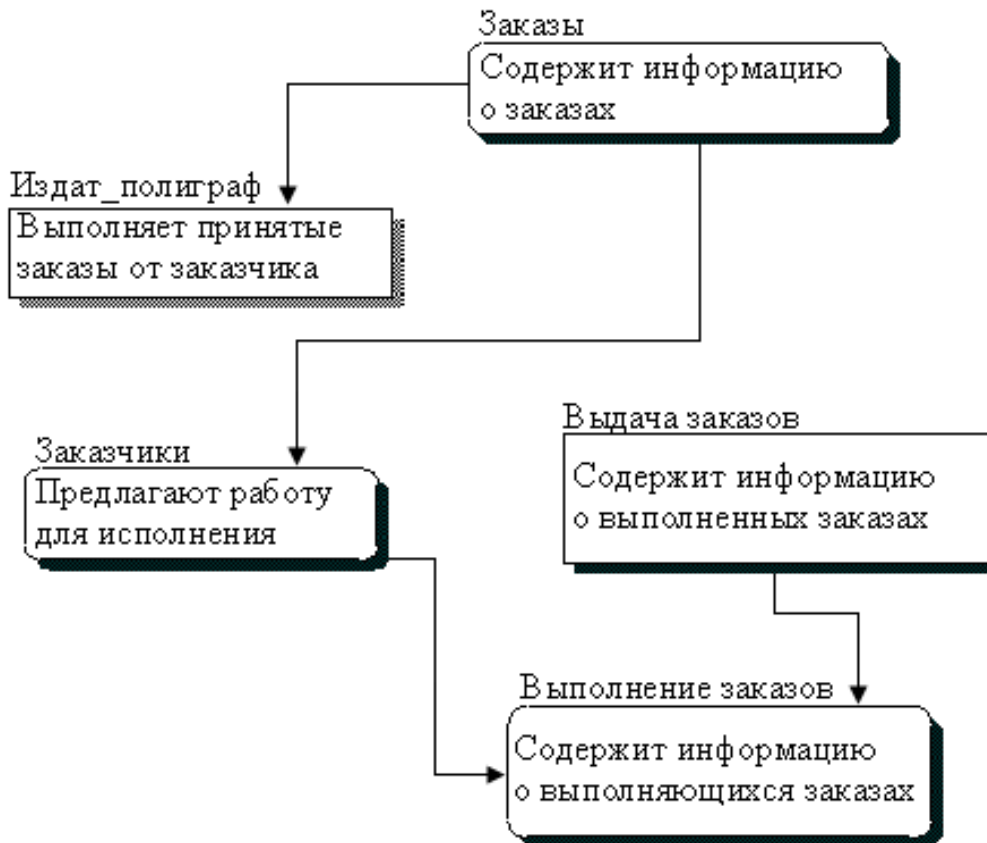


Рис. 7.4. Логическая модель данных на уровне определений

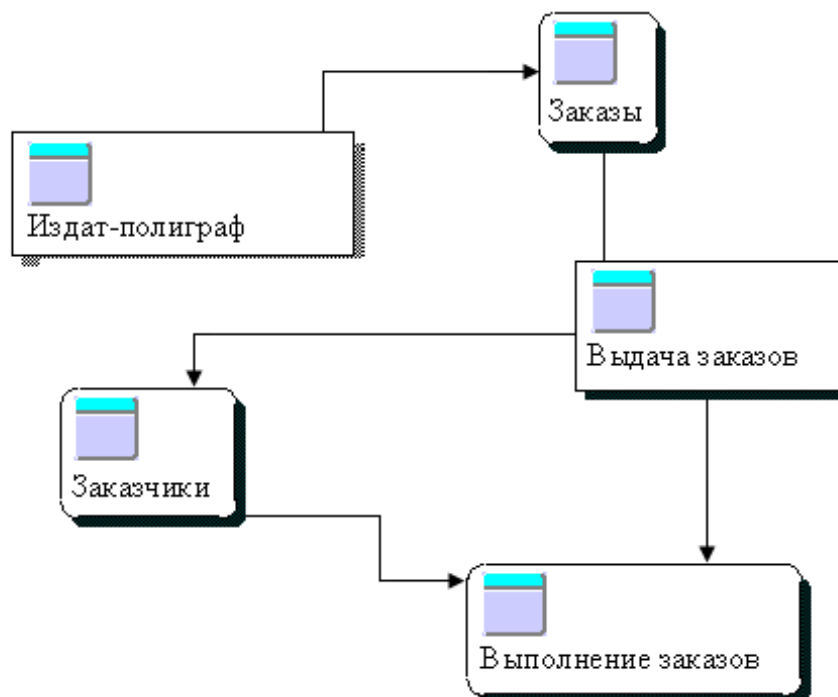


Рис. 7.5. Логическая модель данных на уровне презентаций (иконок)

Примечание. В вышепредставленных уровнях моделей сущность «Выполнение заказов» является ассоциативной и предназначена для разрешения неспецифического отношения $n \times m$ между сущностями «Выдача заказов», «Заказчики».

7.4. Редактор VISIO и логическое моделирование данных

VISIO как инструмент моделирования данных уступает CASE-средству AllFusion ERwin Data Modeler. Тем не менее при отсутствии CASE-средства для этих целей можно воспользоваться рассматриваемым здесь графическим редактором. Выбрав в нем Меню: File → New → Database → Database Model Diagram, можно воспользоваться услугами появившихся трафаретов «Entity Relationship» и «Object Relational», а также секцией главного меню Database, представленным на рис. 7.6.

С применением этих трафаретов и опций секции меню Database можно создать логическую модель данных либо импортировать ERwin ERX-файлы для дальнейшей работы в VISIO.

Выйти на упомянутые выше трафареты и секцию меню можно также, выбрав Меню: File → New → Database → ER Source Model. Правда, в этом случае ряд опций секции Database будет несколько другим.

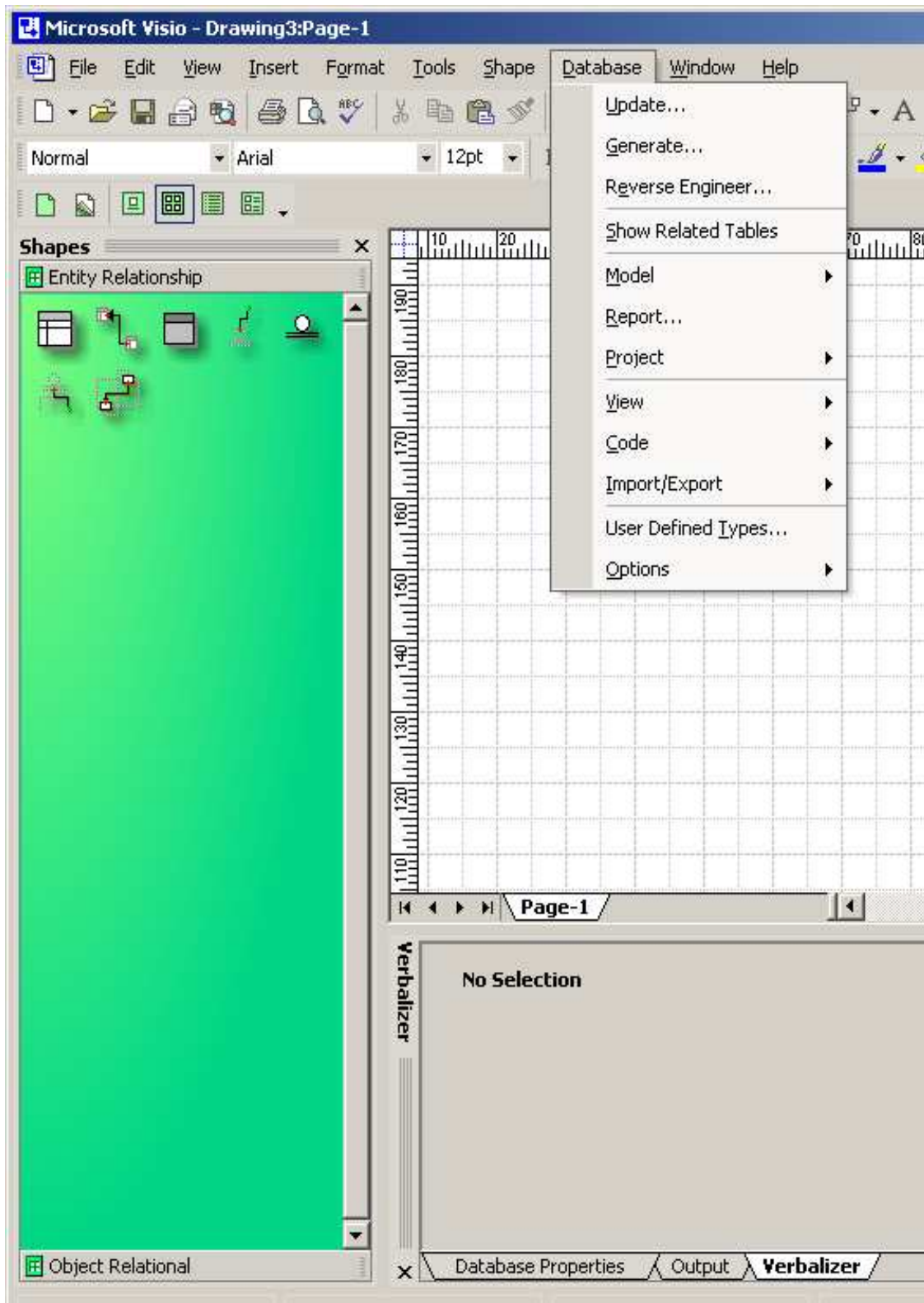


Рис. 7.6. Экран VISIO после выбора Меню:
File → New → Database → Database Model Diagram

Вопросы для самоконтроля

1. Для чего предназначена диаграмма ERD?
2. Перечислите основные элементы ER-диаграммы.
3. Какой инструментарий используется для построения логической модели данных?
4. Назовите преимущества использования CASE-средства AllFusion ERwin Data Modeler при построении логической модели данных.
5. На каких уровнях ER-диаграммы позволяет показать логическую модель данных CASE-средство AllFusion ERwin Data Modeler?
6. Какие нотации ER-диаграммы можно использовать при построении логической модели данных при работе с AllFusion ERwin Data Modeler?

8. ФИЗИЧЕСКОЕ МОДЕЛИРОВАНИЕ ДАННЫХ

8.1. Разработка физической модели данных в AllFusion ERwin Data Modeler

Физический уровень данных – это, по сути, отображение системного каталога, зависящего от реализации СУБД [4, 19].

Различают два уровня физической модели данных:

- 1) *трансформационная модель* (Transformation Model);
- 2) *модель СУБД* (DBMS Model).

Первая модель содержит информацию для реализации отдельного проекта, который может быть частью общей ИС и описывать подмножество предметной области. AllFusion ERwin Data Modeler поддерживает ведение отдельных проектов, позволяя проектировщику выделять подмножество модели в виде предметных областей (Subject Area).

Вторая модель автоматически генерируется из первой и является точным отображением системного каталога СУБД. AllFusion ERwin Data Modeler поддерживает эту модель путем генерации системного каталога.

Ниже приведен список кнопок палитры инструментов, а также операции, используемые при физическом моделировании данных, и последовательность их выполнения.

Кнопки палитры инструментов (Tool Box):

[Select] – установка фокуса на каком-либо объекте модели;

[Independent table] – создание новой таблицы;

[View table] – создание нового представления;

[View relationship] – создание связи между представлением и временной таблицей;

[Identifying relationship] – создание идентифицирующей связи;

[Non-identifying relationship] – создание неидентифицирующей связи.

Операции к физическому моделированию данных и последовательность их выполнения даны в таблице [4, 19].

Таблица

Операции к физическому моделированию данных

| Операция | Рекомендуемая последовательность действий. Примечания |
|--------------------|--|
| 1. Общие установки | Тип модели – Physical |

| 1.1. Задание нотации | Меню: Model → Model Properties → <Вкл. Notations> = {[IDEFIX] [IE] [DM]} |
|--|---|
| Продолжение таблицы | |
| Операция | Рекомендуемая последовательность действий. Примечания |
| 1.2. Переключение между уровнями отображения диаграммы | {Панель инструментов → {[Table View] [Column View] [Comments View]} <Свободное поле диаграммы> → RClick → Display Level → <Необходимый уровень отображения: Table, Column, Primary Key, Comment, Physical Order>} |
| 1.3. Выбор сервера и т. п. | Меню: Database → Choose Database – после выбора СУБД ERwin автоматически создает имена таблиц и колонок на основе имен соответствующих сущностей и атрибутов. Щелкнув правой кнопкой по таблице и выбрав Table Properties или Column, можно задавать или изменять свойства таблиц и колонок |
| 2. Работа с таблицами | Тип модели – Physical; режим отображения – Table |
| 2.1. Внесение новой таблицы | Палитра инструментов: [Table] → <Область в окне диаграммы> → Click |
| 2.2. Задание свойств таблицы (новой либо существующей) | <Выбранная таблица> → RClick → Table Properties → Диалог Tables |
| 2.3. Описание колонок | <Выбранная таблица> → RClick → Columns → Диалог Columns |
| 3. Работа с представлениями | Тип модели – Physical |
| 3.1. Внесение представления | Палитра инструментов: [View] → <Область в окне диаграммы> → Click |
| 3.2. Задание свойств представления | <Выбранное представление> → RClick → Database View Properties → <Диалог Views> |
| 3.3. Установление связи с представлениями | Палитра инструментов: [View Relationship] → Окно диаграммы: <Родительская таблица> → Click → <Представление> → Click |
| 3.4. Редактирование свойств колонок представления | <Выбранное представление> → RClick → Database View Columns |

| 4. Прямое и обратное проектирование | Тип модели – Physical |
|--|---|
| Окончание таблицы | |
| Операция | Рекомендуемая последовательность действий. Примечания |
| 4.1. Генерация системного каталога БД из модели данных | {<Меню> → Tools → Forward Engineer [Forward Engineer]} |
| 4.2. Генерация модели из системного каталога БД | {<Меню> → Tools → Reverse Engineer [Reverse Engineer]} |
| 4.3. Синхронизация системного каталога БД и текущей модели | {<Меню> → Tasks → Complete Compare [Complete Compare]} |

8.2. Построение физической модели данных при разработке программного обеспечения «Информационная система обработки полиграфических заказов»

Используя результаты логического моделирования, строят физическую модель, представленную ниже на различных уровнях:

- на уровне таблиц (рис. 8.1);
- на уровне колонок (рис. 8.2);
- на уровне первичных ключей (рис. 8.3);
- на уровне физического порядка (рис. 8.4).

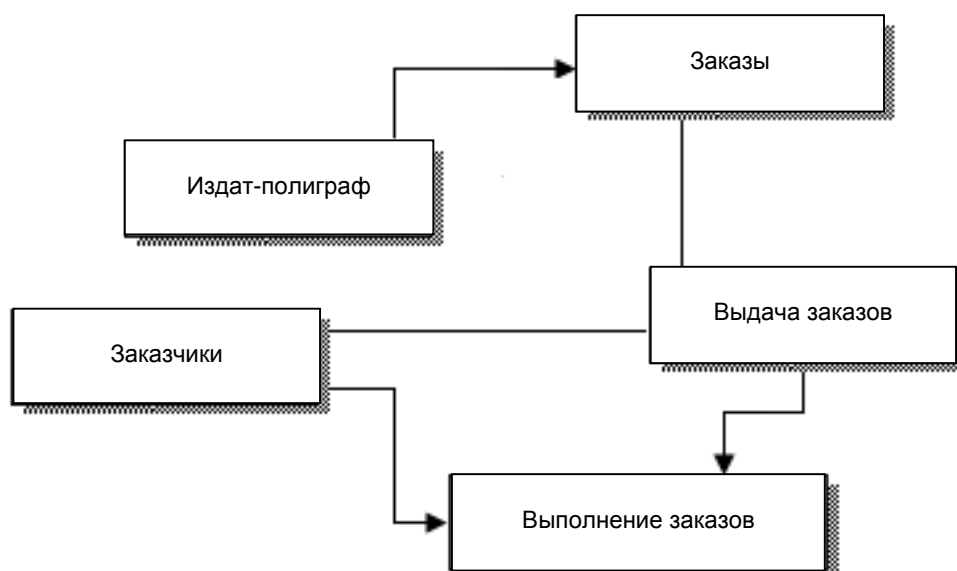


Рис. 8.1. Физическая модель данных на уровне таблиц

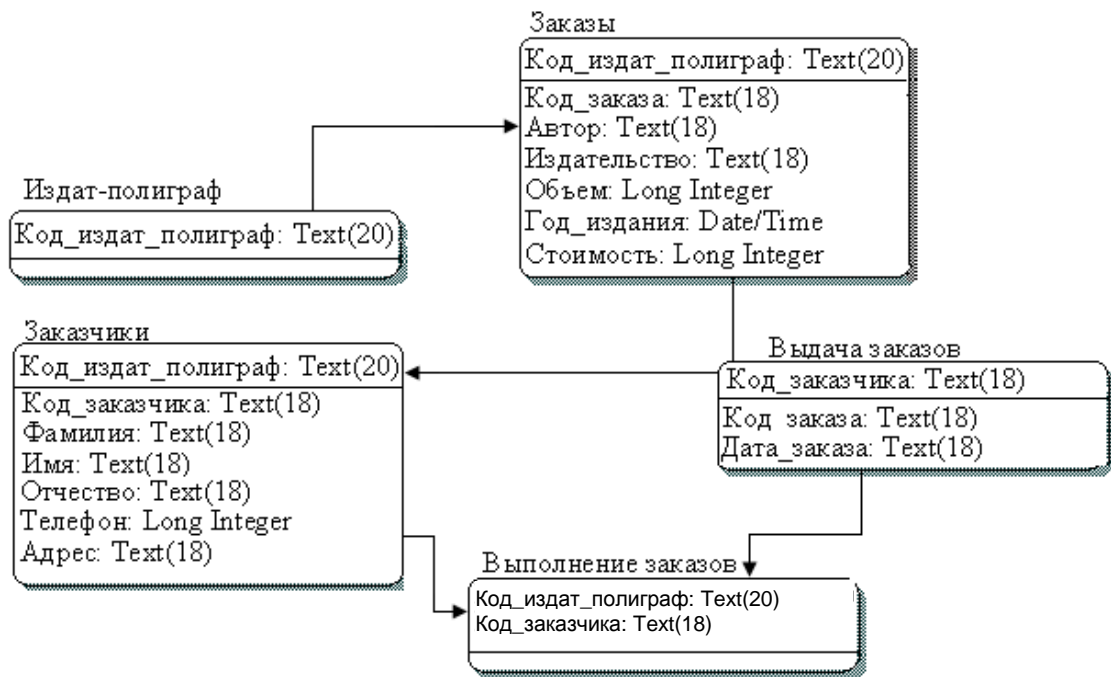


Рис. 8.2. Физическая модель данных на уровне колонок

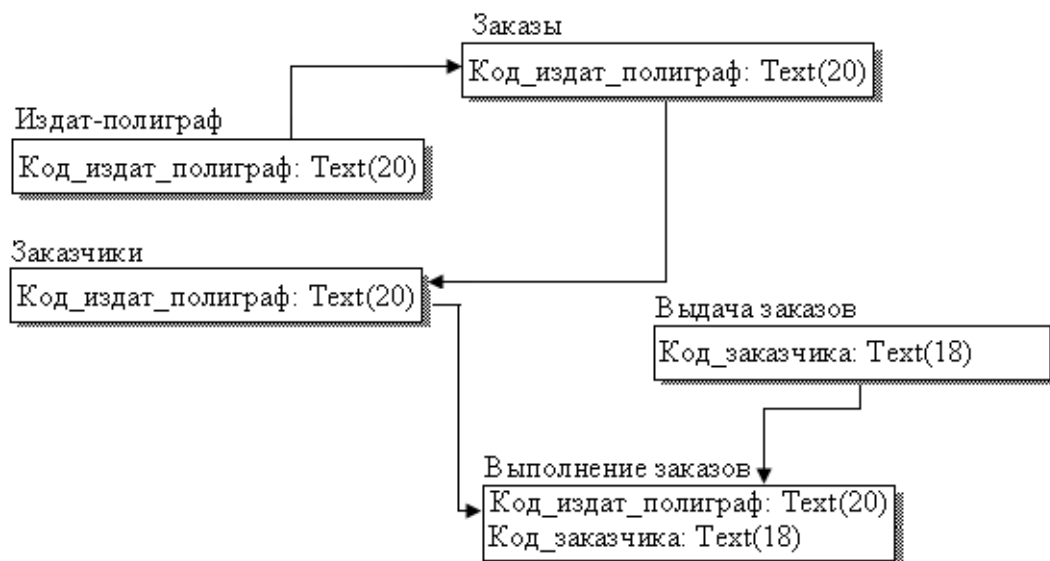


Рис. 8.3. Физическая модель данных на уровне первичных ключей

Выполнив прямое проектирование на основании физической модели, получают системный каталог БД. Воспользовавшись командой Меню: Tools → Forward Engineer, получают системный каталог Access-приложение, структура которого показана на рис. 8.5.

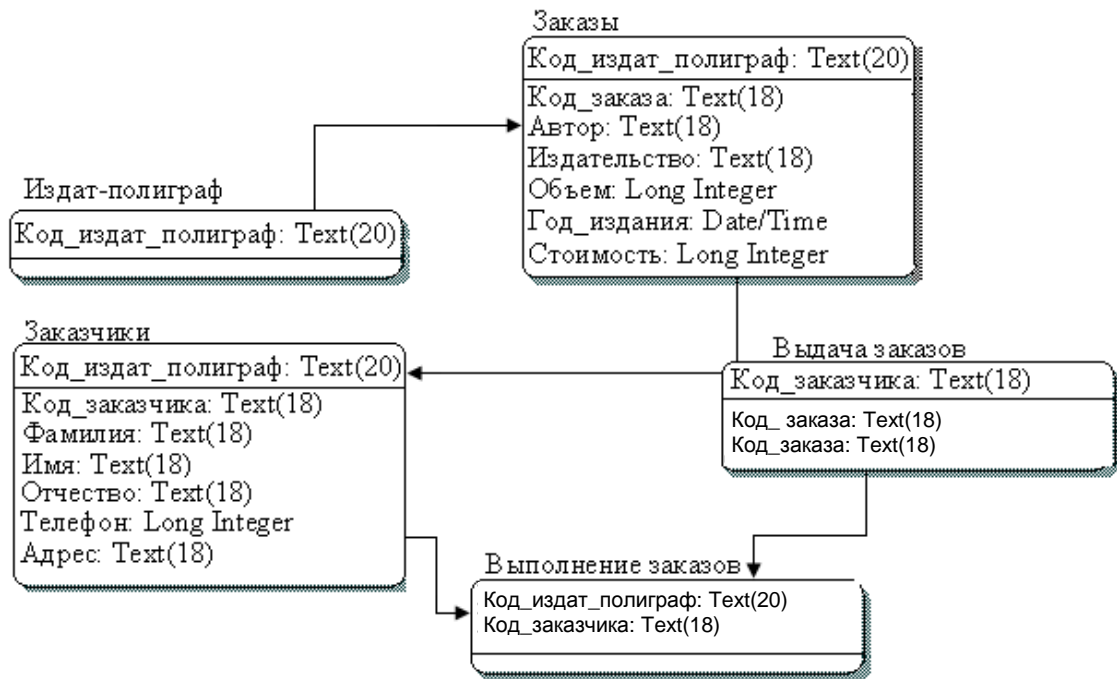


Рис. 8.4. Физическая модель данных на уровне физического порядка

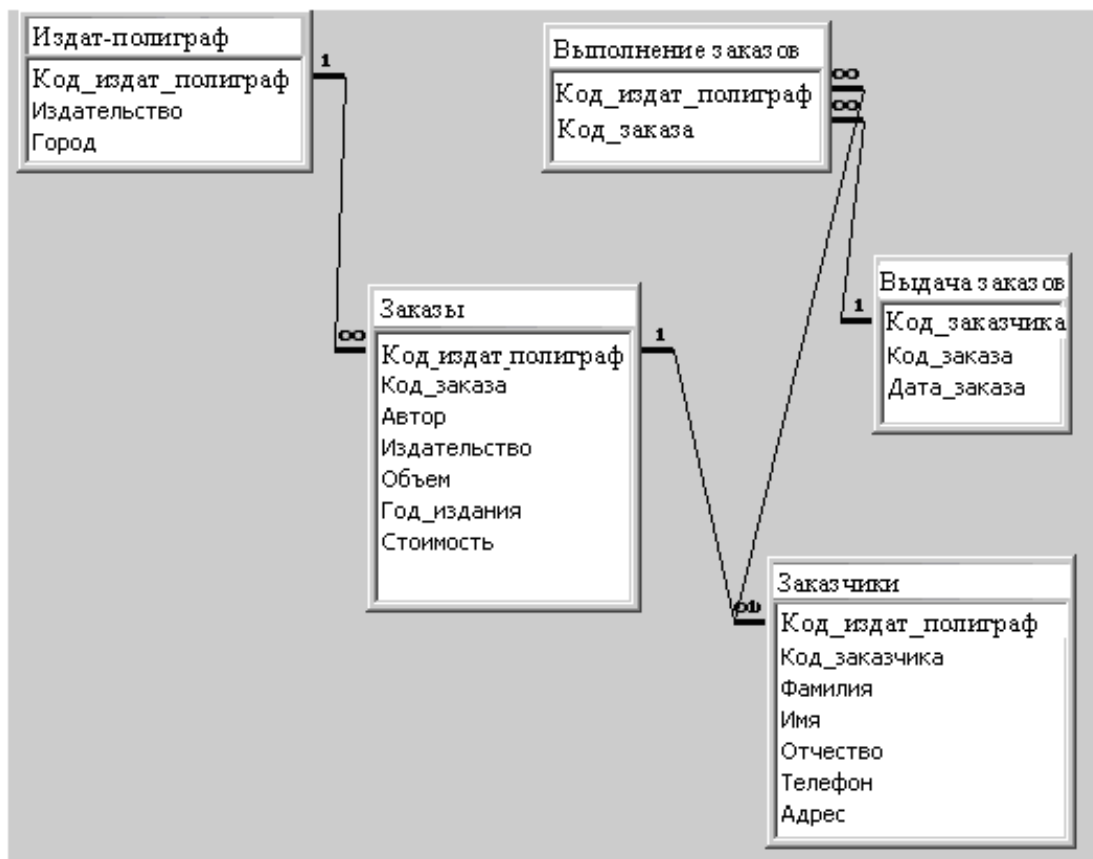


Рис. 8.5. Структура системного каталога Access-приложения

Воспользовавшись вышеупомянутым системным каталогом и выбрав команду Меню: Tools → Reverse Engineer – Select Template, получают модель данных, которая показана ниже на рис. 8.6, но не в нотации IDEF1X, а IE (для разнообразия).

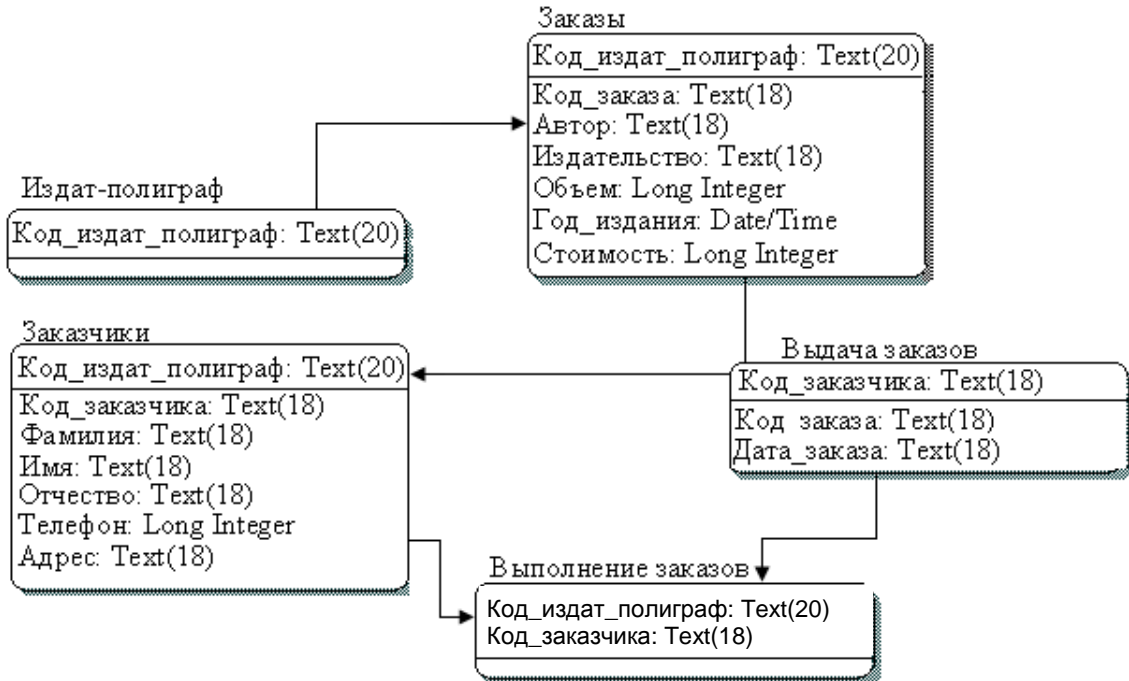


Рис. 8.6. Модель данных в нотации IE

Модель данных в нотации IE может быть использована для последующей модификации исходной БД либо документирования существующей БД, если ее модели по каким-то причинам нет в наличии.

8.3. Редактор VISIO и физическое моделирование данных

VISIO поддерживает также построение физической модели данных по существующей логической модели. Для этого достаточно выбрать Меню: File → New → Options → Document ... Table Data Types [Show Physical].

Для генерации системного каталога либо DLL-скрипта по модели данных можно воспользоваться опцией Generate секции Database, а

для создания модели данных по скрипту либо системному каталогу – опцией Reverse Engineer. В этом смысле VISIO мало уступает CASE-средству AllFusion ERwin Data Modeler.

Вопросы для самоконтроля

1. В чем отличие физической модели данных от логической?
2. Как построить физическую модель данных по логической модели с использованием CASE-средства AllFusion ERwin Data Modeler?
3. Расскажите, как построить физическую модель данных по логической модели с использованием графического редактора VISIO.
4. Как создать системный каталог БД по ее физической модели в AllFusion ERwin Data Modeler, в VISIO?
5. Что такое реинжиниринг и как он реализуется с использованием перечисленных выше средств: AllFusion ERwin Data Modeler, VISIO?

9. ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ ПОДХОД К РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

9.1. Понятия ООА, ООД и ООР

При объектно-ориентированной разработке ПО находят применение объектно-ориентированный анализ (ООА), проектирование (ООД) и программирование (ООР) [9, 11].

1. *ООА (Object Oriented Analysis)* – фаза анализа, при которой требования к системе воспринимаются с точки зрения классов и объектов, выявленных в предметной области.

2. *ООД (Object Oriented Design)* – фаза проектирования, соединяющая в себе процесс объектной декомпозиции и приемы представления логической, физической, а также статической и динамической моделей проектируемой системы, т. е. ООД основывается на объектно-ориентированной декомпозиции; ООД использует многообразие приемов представления моделей, отражающих логические (классы и объекты) и физические (модули и процессы) структуры системы, а также ее статические и динамические аспекты.

3. *ООР (Object Oriented Programming)* – фаза программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром отдельного класса, а классы образуют иерархию наследования, т. е. ООР использует в качестве базовых элементов объекты, а не алгоритмы; каждый объект является экземпляром какого-либо класса; классы организованы иерархически.

9.2. Объектная модель

Выделяют пять стилей программирования, которые различаются иерархически, с присущими им видами абстракций:

1) *процедурно-ориентированный* – основным видом абстракций являются алгоритмы;

2) *объектно-ориентированный* – классы и объекты;

3) *логико-ориентированный* – цели часто выражены в терминах исчисления предикатов;

4) *ориентированный на правилах* – правило «если то»;

5) *ориентированный на ограничения* – инвариантные соотношения.

Каждый стиль программирования имеет свою концептуальную базу. Для объектно-ориентированного стиля это объектная модель (ОМ). Она имеет в своей основе абстрагирование, инкапсуляцию, полиморфизм, наследование, иерархию и модульность.

Абстракция выделяет существенные характеристики некоторого объекта, отличающие его от других видов объектов, и таким образом четко определяет его концептуальные границы с точки зрения наблюдателя. Например, абстракция «файл» требует определения его объема памяти на занимающем устройстве, имеет имя и содержание. Эти свойства данной абстракции являются статическими, т. е. имя, размер, содержание, конкретные значения каждого из перечисленных атрибутов динамичны (изменяются в процессе использования файла).

Инкапсуляция – это объединение данных и методов, предопределяющих связь первых с «внешним миром», т. е. инкапсуляция предохраняет данные объекта от нежелательного доступа, позволяя объекту самому управлять доступом к своим данным.

Полиморфизм означает, что клиент может рассматривать различные объекты как одинаковые и с каждым из объектов будет вести себя соответствующим образом. Например, с точки зрения клиента операция «снятие суммы» с расчетного счета и со сберегательного счета одинакова. Однако фактические реализации этих двух методов могут быть абсолютно разными. Возможность рассматривать различные вещи единообразно, хотя каждый ведет себя по-своему, – суть полиморфизма. При этом клиенты могут оставаться в неведении относительно вопросов, которые их не касаются.

Наследование – это, когда, имея некоторый объект, можно создавать новый, автоматически поддерживающий все или некоторые возможности (свойства) старого.

Иерархия – упорядочение абстракций, т. е. расположение их по уровням. Основными видами иерархических структур сложных программных систем являются структура классов и структура объектов. Первая определяет общность структур (взаимосвязь абстракций) и поведение внутри системы. Вторая показывает основные механизмы взаимодействия объектов друг с другом.

Модуль – отдельно компилируемый фрагмент программы. В языке C++, Pascal модуль представляет собой самостоятельную языковую конструкцию. В этих языках классы и объекты составляют

логическую структуру ПО, а модули образуют физическую структуру системы. Рекомендуется в отдельные модули помещать логически связанные классы и объекты. В процессе разбиения системы на модули могут быть полезны два правила:

1. При распределении классов и объектов по модулям необходимо учитывать возможность их повторного использования.

2. Поскольку многие компиляторы создают отдельный сегмент для каждого модуля, могут появиться ограничения на его размер.

Преимущества объектной модели:

– ОМ позволяет в полной мере использовать выразительные средства объектно-ориентированного языка программирования;

– ОМ существенно повышает уровень унификации разработки и пригодность для повторного использования не только программ, но и проектов;

– использование ОМ приводит к построению систем на основе стабильных промежуточных описаний;

– ОМ уменьшает риск разработки сложных систем.

Недостатки объектной модели:

– некоторое замедление работы программ. Одно обращение к методу занимает в 1,7–2,5 раза больше времени;

– динамическое размещение и уничтожение объектов нежелательно для решения задач с ограниченными ресурсами времени;

– увеличение начальных затрат на разработку системы и общего времени на проектирование по сравнению со структурным подходом;

– затруднения, связанные с новым типом мышления при использовании объектного подхода.

9.3. Объектно-ориентированный процесс разработки программного обеспечения

Известно много схем объектно-ориентированной разработки ПО:

– Objectory;

– Rational Objectory Process (ROP) (на базе предыдущего);

– Unified Process-Up (на базе Objectory);

– Rational Unified Process (RUP) (модификация UP);

– Extreme Programming (XP);

– Iconex (среднее между RUP и XP) и др.

Ниже на рисунке приведена упрощенная схема процесса разработки ПО, согласующаяся с Objectory [14].

Это интерактивный и пошаговый процесс, при котором ПО разрабатывается и реализуется по частям.

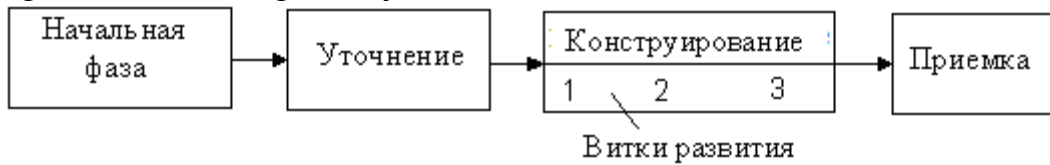


Рисунок. Упрощенная схема процесса разработки ПО

Фаза «Конструирование» состоит из многих итераций (1, 2, ...). Каждая итерация содержит все обычные фазы ЖЦ ПО: анализ, проектирование, реализацию и тестирование.

Что касается фаз, предшествующих конструированию, то:

- в начальной фазе пишется экономическое обоснование и определяются границы проекта (варианты использования);

- в фазе «Уточнение» требования выявляются более детально (детализируются варианты использования), выполняется высокоуровневый анализ и проектирование для построения базовой архитектуры и создается план для конструирования (какие варианты использования и на каких итерациях конструирования должны быть реализованы);

- фаза «Приемка» может включать бета-тестирование, оптимизацию производительности и обучение пользователей.

Из вышесказанного видно, что этапы анализа и проектирования имеют место как в фазе «Уточнение», так и на каждой итерации фазы «Конструирование». Поэтому в дальнейшем акцентируем свое внимание на диаграммных техниках поддержки ООА и ООД.

9.4. Введение в UML и инструментарий его поддержки

Что касается нотации, то следует отдать предпочтение унифицированному языку моделирования UML (Unified Modeling Language).

Создание UML началось в 1994 г., когда Буч и Рамбо предприняли попытку объединить свои методы ООА и проектирования ПО.

В 1995 г. к ним присоединился Якобсон [12, 13]. Первая версия UML

появилась в 1997 г., а версия UML 1.1 – в конце 1997 г. и была утверждена и принята на вооружение всеми крупнейшими производителями ПО.

Создатели UML представляют его как язык определения, представления, проектирования и документирования программных систем различной природы.

С позиций объектно-ориентированного подхода интерес представляет пакет Paradigm Plus фирмы Computer Associates и Rational Rose фирмы Rational Software.

Эти инструменты позволяют строить объектную модель в различных нотациях (OMT, UML, Буч) и генерировать на основе полученной модели приложение на C++, Visual Basic, Power Builder, Java, Ada, SmallTalk и др.

9.5. Инструментальная среда Rational Rose

Общая характеристика Rational Rose. Это CASE-средство представляет собой современное и мощное средство анализа, моделирования и разработки программных систем.

В Rational Rose UML стал базовой технологией визуализации и разработки программ, что определило популярность и стратегическую перспективность этого инструментария.

В версии Rational Rose аккумулированы все современные достижения в области информационных технологий:

- интеграция с Microsoft Visual Studio 6, что включает в себя поддержку на уровне прямой и обратной генерации кодов и диаграмм Visual Basic 6, Visual C++ 6, Visual J++ 6;

- непосредственная работа (инжиниринг и реинжиниринг) с исполняемыми модулями и библиотеками форматов exe, dll, tlb, ocx;

- поддержка MSTS (Microsoft Transaction Server) и ADO (ActiveX Data Object) на уровне шаблонов и исходного кода, а также элементов стратегической технологии Microsoft COM+ (DCOM);

- полная поддержка CORBA 2.2, включая реализацию, технологию компонентной разработки приложений с БД, языка определения интерфейса IDL и языка определения данных DDL;

- полная поддержка Java-приложений JDK 1.2, включая прямую и обратную генерацию классов Java формата JAR, а также работу с файлами форматов CAB и ZIP.

Модель Rational Rose поддерживает четыре представления (Views):

- 1) представление вариантов использования (ВИ);

- 2) логическое представление;
- 3) представление компонентов;
- 4) представление размещения.

Особенности работы интерфейса Rational Rose. Интерфейс Rational Rose состоит из различных интерфейсов, основными из которых являются:

- *главное меню* (отдельные пункты объединяют сходные операции, относящиеся ко всему проекту в целом);
- *стандартная панель инструментов* (вид ее определяется разрабатываемой диаграммой, выбранной нотацией и настройкой);
- *окно браузера* (упрощает навигацию и позволяет отыскивать любой элемент модели в проекции);
- *специальная панель инструментов* (по умолчанию предлагается для построения диаграмм классов);
- *окно диаграммы* (необходимо для визуализации различных представлений модели проекта);
- *окно документации* (предназначено для документирования элементов представления модели);
- *окно журнала – Log* (необходимо для автоматической записи различной служебной информации, образующейся в ходе работы с программой).

Начало работы над проектом в среде Rational Rose. Для создания нового проекта в основном меню следует выбрать команду File → New (если установлен мастер типовых проектов, то можно воспользоваться его услугами). При наличии готового проекта (.mdl-файл) его можно открыть, выбрав File → Open.

В конце сеанса осуществляется сохранение при помощи команды File → Save, или File → Save As.

Имеется возможность настроить глобальные параметры среды (Tools Options).

Для изменения цвета линий необходимо воспользоваться командой Edit → Diagram → Object → Properties → Line → Color.

Общий процесс работы над проектом заключается в добавлении на диаграммы соответствующих графических элементов, установлении отношений между этими элементами, их спецификации и документировании. После проверки правильности модели и согласованности ее элементов можно сгенерировать текст программного кода на один из выбранных языков программирования.

Следует быть осторожными при добавлении элементов на диаграммы, так как они заносятся в браузер, а при удалении – из браузера автоматически не удаляются. Поэтому необходимо предпринять дополнительные меры для удаления ненужного элемента из браузера (для этого следует выбрать этот элемент и нажать Ctrl+D).

Вопросы для самоконтроля

1. Поясните смысл понятий ООА, ООД и ООР.
2. Назовите известные вам стили программирования.
3. Что собой представляет объектная модель?
4. Перечислите достоинства и недостатки объектной модели.
5. Назовите способы идентификации классов.
6. Перечислите основные этапы объектно-ориентированного процесса разработки ПО.
7. Что представляет собой унифицированный язык моделирования UML?
8. Какие другие методы построения моделей ПО вы знаете?

10. МОДЕЛЬ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ И ЕЕ РАЗРАБОТКА

10.1. Элементы нотации диаграммы прецедентов и особенности ее построения

Модель вариантов использования включает в себя список действующих лиц (актеров) с описанием их ролей, список вариантов использования (прецедентов), диаграмму вариантов использования и описание вариантов использования [10, 12–14].

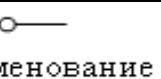
Что касается диаграмм вариантов использования (прецедентов), то они являются исходным концептуальным представлением системы на этапе ООА. Диаграммы вариантов использования позволяют:

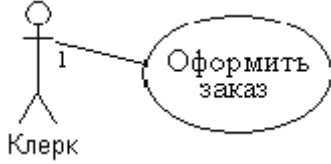

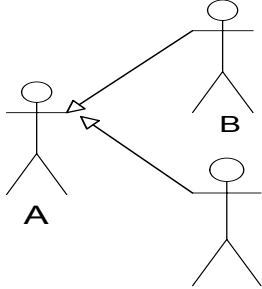
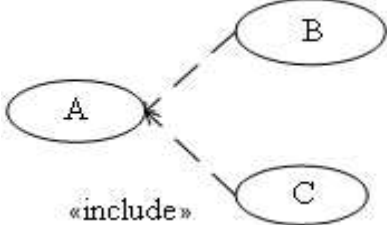

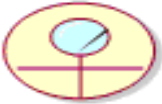

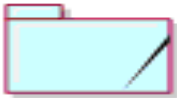
- определить общие границы и контекст моделируемого ПО;
- сформулировать общие требования к функциональному поведению создаваемой системы;
- разрабатывать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей;
- подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Основные элементы и понятия, связанные с диаграммой вариантов использования (прецедентов), представлены в табл. 10.1.

Таблица 10.1



Основные элементы диаграммы вариантов использования

| Наименование | Графическое представление |
|---|--|
| Вариант использования (прецедент, Use Case) |  |
| Действующее лицо (Actor) |  |
| Интерфейс |  |

| Наименование | Графическое представление |
|--|--|
| Отношение ассоциации |  |
| Отношение расширения |  |
| Отношение обобщения |  |
| Отношение включения |  |
| Дополнительные значки для моделирования бизнес-процессов | |
| Бизнес-актер (Business actor) |  |
| Бизнес-работник (Business worker) |  |
| Бизнес-сущность (Business entity) |  |
| Организационное подразделение (Organization Unit) |  |

| | |
|---------------------|--|
| (Organization Unit) | |
|---------------------|--|

Окончание табл. 10.1

| Наименование | Графическое представление |
|---|--|
| Бизнес-процесс (Business Use Case) |  |
| Реализация бизнес-процесса (Business Use Case Realization) |  |

Вариант использования (прецедент, Use Case) определяет законченный аспект или фрагмент поведения некоторой сущности без раскрытия внутренней структуры этой сущности, т. е. характеризует способ применения этой сущности пользователем (актером).

Действующее лицо (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.

Интерфейс служит для спецификации параметров модели, которые видны извне без указания их внутренней структуры.

Отношение ассоциации обозначает роль актера в отдельном варианте использования.

Отношение расширения иллюстрирует случай, когда прецедент А может быть расширен по запросу прецедентом В.

Отношение обобщения показывает, что актер А является предком актеров В и С.

Отношение включения демонстрирует, что прецедент А включает возможности прецедентов В и С.

Бизнес-актер (Business actor) представляет исполнителя бизнес-функций, связанного с работой системы.

Бизнес-работник (Business worker) означает работника внутри системы, выполняющего некоторые бизнес-функции.

Бизнес-сущность (Business entity) – пассивный бизнес-объект, не инициирующий взаимодействие.

Организационное подразделение (Organization Unit) представляет пакет, содержащий другие бизнес-объекты.

Бизнес-процесс (Business Use Case) означает отдельный бизнес-процесс.

Реализация бизнес-процесс (Business Use Case Realization) представляет реализацию отдельного бизнес-процесса.

10.2. Разработка диаграммы прецедентов в Rational Rose

Rational Rose – наиболее удобный инструмент для построения диаграмм вариантов использования.

Единственным отступлением от синтаксиса UML является то, что наименование прецедентов размещается при работе с ним не в овале варианта использования, а под ним. Это, на наш взгляд, улучшает внешний вид диаграммы.

Операции к построению диаграмм вариантов использования (диаграмм прецедентов) и последовательность их выполнения даны в табл. 10.2 [15, 17, 19].

Таблица 10.2

Операции к построению диаграмм вариантов использования

| Операция | Рекомендуемая последовательность действий. Примечания |
|---|---|
| 1. Создание действующего лица | Окно Browser: Use Case View → RClick → New → Actor → New Class ← = <Требуемое имя> |
| 2. Документирование действующего лица | Окно Browser: <Требуемый элемент> → Окно Documentation:= <Текстовое описание действующего лица> |
| 3. Создание варианта использования | Окно Browser: Use Case View → RClick → New → Use Case → New Class ← = <Требуемое имя> |
| 4. Документирование варианта использования | Окно Browser: <Требуемый элемент> → Окно Documentation: = <Текстовое описание варианта использования> |
| 5. Привязка дос-файла с описанием варианта использования к варианту использования | Окно Browser: <Требуемый прецедент> → RClick → Open Specification → Use Case Specification → Files → <Область окна> → RClick → Insert File → <Требуемый файл> → [Open] → [OK] |
| 6. Создание основной диаграммы вариантов использования | Окно Browser: Use Case View → Main → DbClick – отбуксировать все действующие лица и варианты использования в окно диаграммы |

| | |
|--|---|
| 7. Создание коммуникативной ассоциации | Панель Diagram: {[Association] [Unidirectional Association]} → Окно диаграммы: <Действующее лицо> → LClick_∨ → <Вариант использования> → LClick_^ |
|--|---|

Окончание табл. 10.2

| Операция | Рекомендуемая последовательность действий. Примечания |
|--|---|
| 8. Снабжение ассоциации необязательной меткой стереотипа communicate | Окно диаграммы: <Линия ассоциации> → DblClick → Association Specification → Список Stereotype → Communicate → [OK] |
| 9. Создание включающей связи | Панель Diagram: [Dependency] → Окно диаграммы: <Базовый вариант использования> → LClick_∨ → <Включаемый вариант использования> → LClick_^ → <Линия связи> → DblClick → Dependency Specification → Список Stereotype → Include → [OK] |
| 10. Создание расширяющей связи | Панель Diagram: [Dependency] → Окно диаграммы: <Расширяющий вариант использования> → LClick_∨ → <Базовый вариант использования> → LClick_^ → <Линия связи> → DblClick → Dependency Specification → Список Stereotype → Extend → [OK] |
| 11. Создание дополнительной диаграммы вариантов использования | Окно Browser: Use Case View → RClick → New → Use Case Diagram → New Diagram ← = <Требуемое имя> → DblClick – откроется окно дополнительной диаграммы вариантов использования, в которое можно вносить необходимые элементы |
| 12. Создание диаграммы реализации вариантов использования | Окно Browser: Logical View → RClick → New → Use Case Diagram → New diagram ← = <Требуемое имя> |
| 13. Создание реализации варианта использования | Окно Browser: <Значок диаграммы реализации ВИ> → DblClick → Панель Diagram: [Use Case] → Окно диаграммы: Click → New Use Case → DblClick → Use Case Specification → Name = <Имя, соответствующее в Use Case View> → <Из Stereotype выбрать Use Case Realization> → [OK] |

Для моделирования бизнес-процессов Rational Rose оставляет шесть дополнительных значков:

– *Business actor* (бизнес-актер) – действующее лицо, связанное с

бизнес-системой, например пользователь;

– *Business worker* (бизнес-работник) – работник внутри бизнес-системы, например клерк;

– *Business entity* (бизнес-сущность) – пассивный бизнес-объект, например счет;

– *Organization Unit* (организационное подразделение) – пакет, содержащий другие бизнес-объекты, в том числе и организационные подразделения;

– *Business Use Case* (бизнес-процесс) – отдельный бизнес-процесс;

– *Business Use Case Realization* (реализация бизнес-процесса) – реализация отдельных бизнес-процессов.

Для включения этих значков в линейку инструментов необходимо выбрать панель Diagram: RClick → Customise.

10.3. Создание модели прецедентов при разработке программного обеспечения «Информационная система обработки полиграфических заказов»

Модель вариантов использования.

Действующие лица:

1. Оператор типографии назначает действие заказу.
2. Полиграфическая система – система обслуживания заказа.
3. Администратор типографии – контроль базы данных за поступающими заказами.
4. Служащий типографии оформляет заказы на определенную печать.

Варианты использования.

1. Ввести новый заказ на типоофсетном печатном аппарате.
2. Изменить существующий заказ.
3. Выполнить заказ офсетной печатью.
4. Обновить базу данных типографии.
5. Оформить заказ на глубокую печать.
6. Отклонить заказ на рулонных машинах.

Диаграмма вариантов использования (рис. 10.1):

1. Оператор выбирает пункт «Новый заказ на типоофсетном печатном аппарате» из имеющегося меню.
2. Система выводит форму «Подробности заказа».
3. Оператор вводит наименование заказа, заказчика и то, что заказано.
4. Оператор сохраняет заказ.

5. Система создает новый заказ и сохраняет его в базе данных.

Описание варианта использования «Обновить базу данных типографии».

Назначение – обновление базы данных типографии по мере поступления заказов.

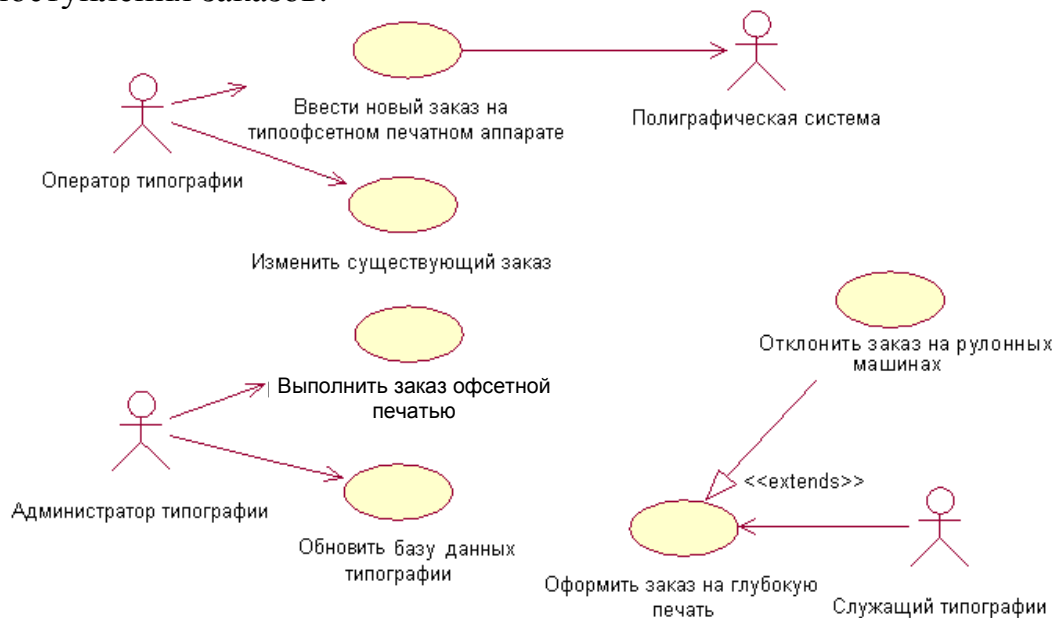


Рис. 10.1. Диаграмма вариантов использования

Основной поток.

1. Исполнитель просматривает заказы с помощью прецедента «Предоставление доступа к заказам».

2. Исполнитель оценивает результаты выполнения заказа.

3. При необходимости исполнитель обновляет базу данных заказов о ходе выполнения заказа.

4. При положительных результатах выполнения заказа исполнитель не меняет стратегии работы.

5. Исполнитель вносит замечания администратора в базу данных.

Альтернативный поток – отсутствует.

Предусловие – успешное завершение прецедента «Предоставление доступа к заказам».

Постусловие – обновилась база данных полиграфических заказов.

10.4. Редактор VISIO и диаграмма вариантов использования

VISIO как инструмент построения диаграмм вариантов использования (прецедентов), пожалуй, несколько уступает Rational Rose. Тем не менее, выбрав в VISIO Меню: File → New → Software → UML Model Diagram, получают возможность воспользоваться трафаретами для построения всех видов UML-диаграмм, секцией UML главного меню, а также окном Model Explorer, показанным на рис. 10.2. Один из упомянутых трафаретов – трафарет «UML Use CASE» представлен на рис. 10.3.

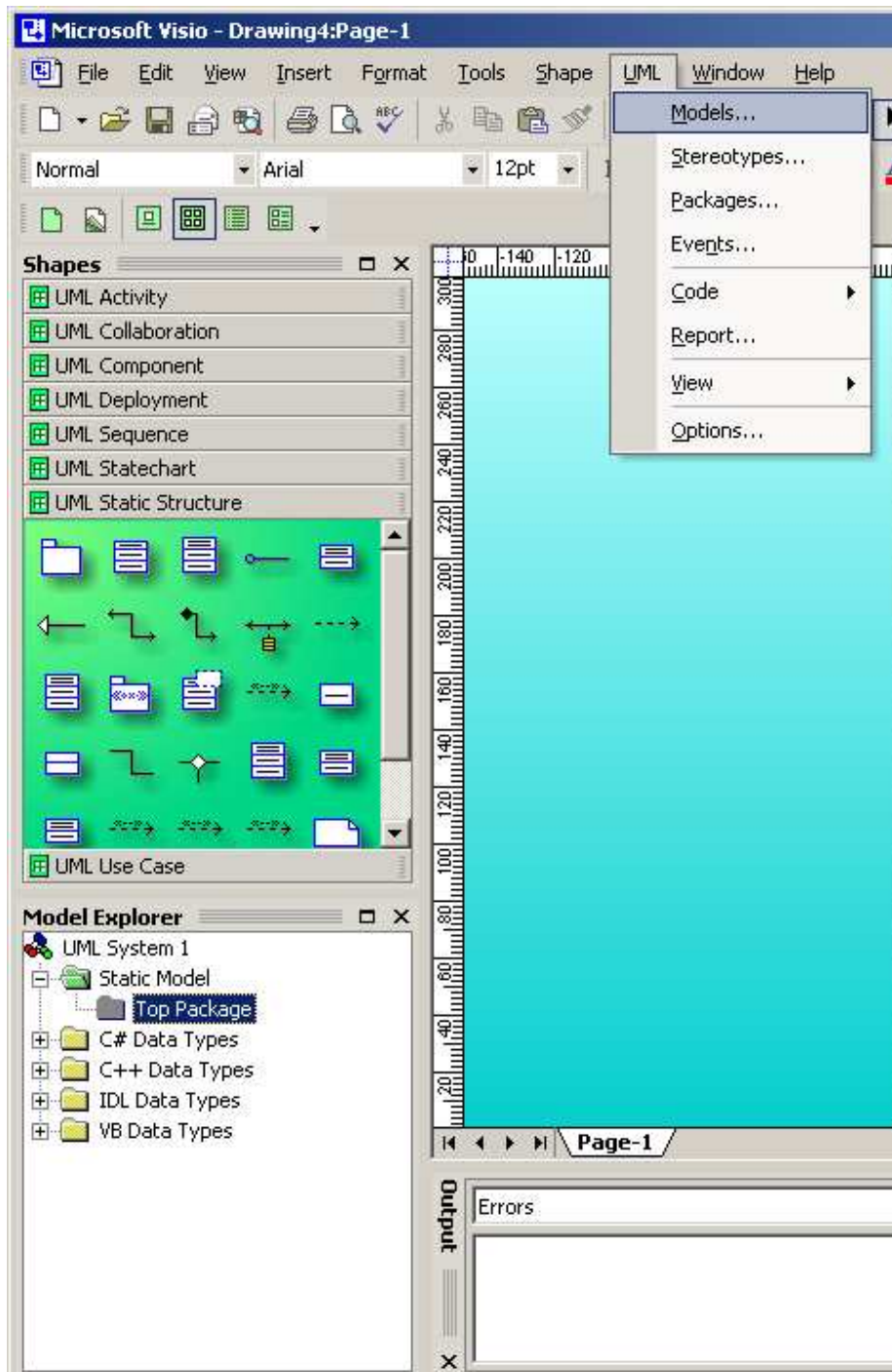


Рис. 10.2. Окно программы после выбора VISIO Меню:
File → New → Software → UML Model Diagram

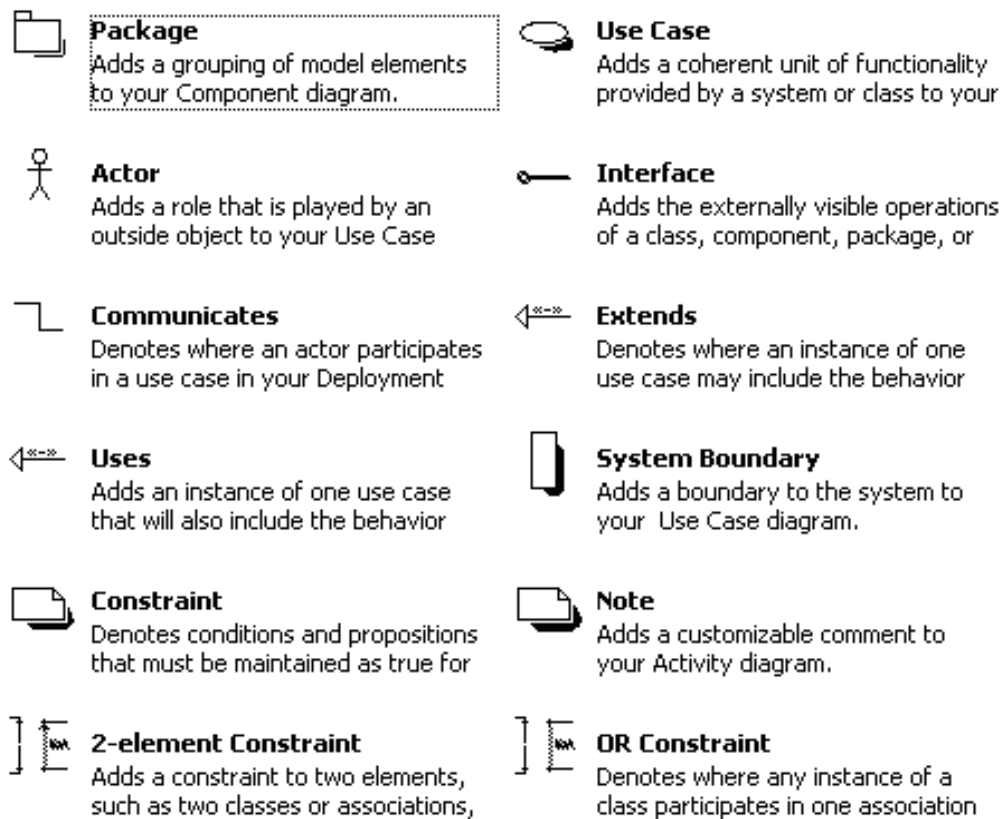


Рис. 10.3. Общий вид трафарета «UML Use CASE»

Как видно, в этом шаблоне набор стрелок для представления отношений между элементами диаграммы вариантов использования не велик. Приходится в основном манипулировать стереотипами. Зато, в отличие от Rational Rose, в трафарете VISIO имеется шаблон System Boundary, позволяющий очертить границы системы. Это иногда весьма существенно.

Вопросы для самоконтроля

1. Для чего используется диаграмма вариантов использования (прецедентов) Use Case?
2. Назовите основные элементы диаграммы Use Case.
3. Какие пункты как минимум должно включать описание варианта использования (прецедента)?
4. Расскажите, как создать диаграмму Use Case в Rational Rose.
5. Как создать Use Case-диаграмму в VISIO?
6. Какие другие средства создания диаграммы вариантов использования вы знаете?

11. ДАЛЬНЕЙШЕЕ РАЗВИТИЕ МОДЕЛИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

11.1. Диаграмма состояний и ее разработка





Диаграмма состояний используется для отдельных классов с целью описания поведения порождаемых ими объектов [10, 12–14].

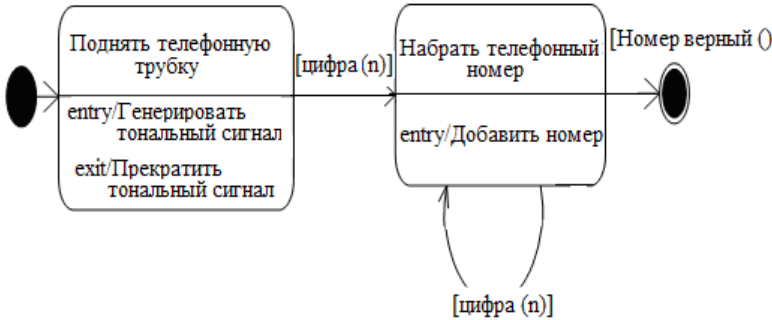
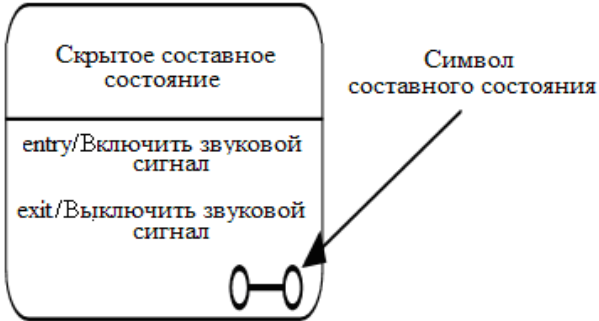
Диаграмма состояний, по существу, является графом специального вида, который представляет некоторый автомат, и, следовательно, для построения и понимания семантики конкретной диаграммы состояний надо учитывать не только особенности представления моделируемой сущности, но и знать общие сведения по теории автоматов.

Автомат UML представляет собой некоторый формализм для поведения элементов моделей и системы в целом. Основные элементы и понятия, связанные с диаграммой состояний, представлены в табл. 11.1.

Таблица 11.1

Основные элементы диаграммы состояний

| Наименование | Графическое представление |
|---------------------|---|
| Состояние |  |
| Начальное состояние |  |
| Конечное состояние |  |
| Переход | <p>Установить телефонное соединение (телефонный номер) [телефонное соединение установлено]</p>  <p>Закончить загрузку почты [почтовый ящик на сервере пуст]/ разорвать телефонное соединение (телефонный номер)</p> |

| Наименование | Графическое представление |
|-----------------------------|--|
| Составное состояние | <p style="text-align: center;">Дозвон до абонента</p>  |
| Скрытое составное состояние |  |

В UML под *состоянием* понимают абстрактный класс, используемый для моделирования отдельной ситуации, в течение которой имеет место выполнение некоторого условия.

Начальное состояние представляет собой частный случай состояния, которое не содержит никаких внутренних действий. В этом состоянии находится объект по умолчанию в начальный момент времени.

Конечное состояние представляет собой частный случай состояния, которое не содержит никаких внутренних действий. В этом состоянии находится объект по умолчанию в конечный момент времени.

Переход – отношения между двумя последовательными состояниями, которые указывают на факт смены одного состояния другим. На диаграмме состояний переход обозначается сплошной линией со стрелкой, направленной в целевое состояние.

Триггерный переход обозначается текстом вида:

<Сигнатура события> '['<Сторожевое условие>']' <Выражение действия>, где *сигнатура события* – событие, которое вызывает

переход; *сторожевое условие* – условие перехода; *выражение действий* – действия, которые выполняются до перехода.

Составное состояние – это такое состояние, которое состоит из других вложенных в него состояний (предсостояний).

Скрытое составное состояние позволяет скрыть внутреннюю структуру составного состояния и улучшить читабельность диаграммы.

11.2. Разработка диаграммы состояний в Rational Rose

В Rational Rose весьма прост процесс построения диаграмм состояний. Имеется возможность замены вложенных состояний субдиаграммами. Операции к построению диаграмм состояний и последовательность выполнения каждой из них даны в табл. 11.2 [15, 17, 19].

Таблица 11.2

Операции к построению диаграмм состояний

| Операция | Рекомендуемая последовательность действий. Примечания |
|--|--|
| 1. Создание диаграммы состояний | Окно Browser: <Требуемый класс> → RClick → New Statechart Diagram → New Diagram ← = <Требуемое имя> → DblClick |
| 2. Создание состояния | Окно Browser: <Значок диаграммы состояний> → DblClick → <Панель Diagram> → [State] → Окно диаграммы → Click – появится элемент состояния, выбрав который можно ввести его имя |
| 3. Создание перехода состояния | Панель Diagram: [State Transition] → Окно диаграммы: <Состояние-источник> → Click_↵ → <Состояние-приемник> → Click_^ |
| 4. Задание именованного события переходу | Поле диаграммы: <Стрелка перехода> → DblClick → State Transition Specification → General → Event = <Имя события> → [OK] |
| 5. Создание исходного состояния | Панель Diagram: [Start State] → Окно диаграммы: Click → Панель Diagram: [State Transition] → Окно диаграммы: <Символ исходного состояния> → Click_↵ → <Состояние-приемник> → Click_^ |
| 6. Создание завершающего состояния | Панель Diagram: [End State] → Окно диаграммы: Click → Панель Diagram: [State Transition] → Окно диаграммы: <Состояние-источник> → Click_↵ → <Завершающее состояние> → Click_^ |

| | |
|--|--|
| 7. Отображение данных перехода состояния | Окно диаграммы: <Стрелка перехода> → RClick → Open Specification → State Transition Specification → Detail → {Guard Condition = <описание> ! Action = <описание> ! Send Event = <описание>} → [OK] |
|--|--|

Окончание табл. 11.2

| Операция | Рекомендуемая последовательность действий. Примечания |
|--|---|
| 8. Определение входящих, исходящих и внутренних действий | Окно диаграммы: <Значок состояния> → RClick → Open Specification → State Specification → Actions → <Поле закладки> → RClick → Insert → Entry → DblClick → Action Specification → When-список → {On Entry On Exit Do On Event} → <Список Type> → {Action Send Event} → <Name> = <Имя> → 0{Send arguments = <описание> → Send Target = <описание>}1 → [OK] → [OK] |

Построение диаграммы состояний при разработке ПО «Информационная система обработки полиграфических заказов». Создание диаграммы состояний Statechart Diagram для класса «Заказ растрового клише» представлено на рис. 11.1.

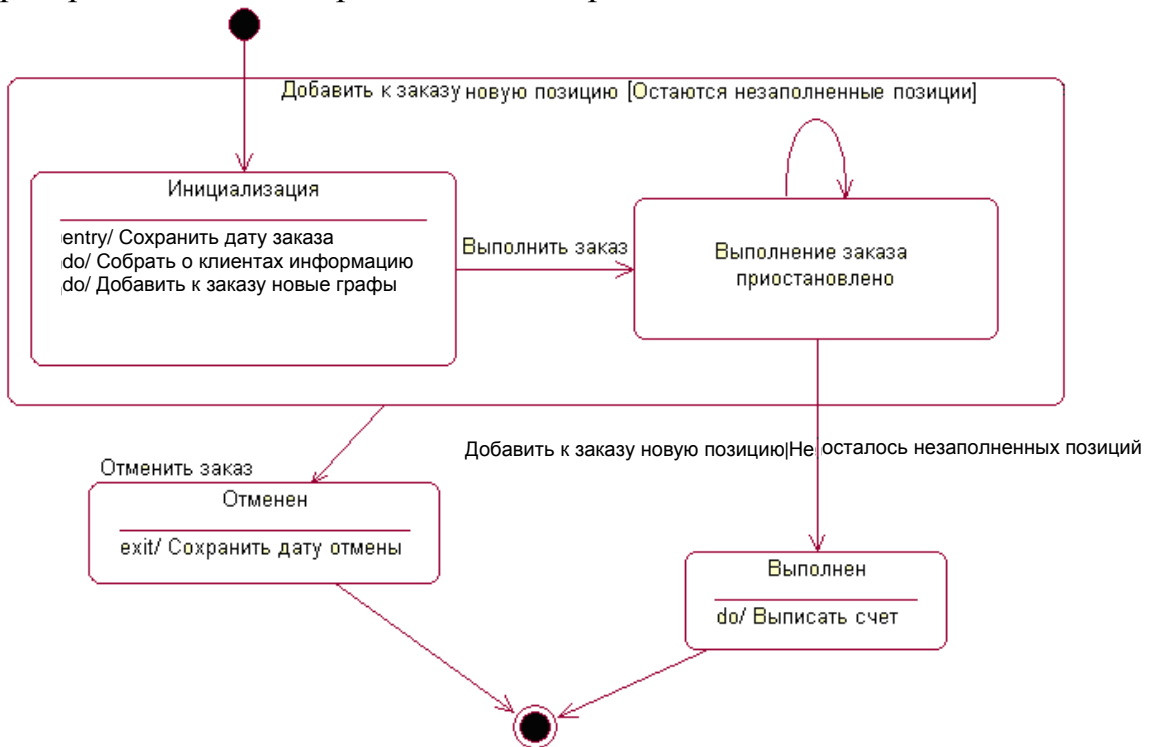


Рис. 11.1. Диаграмма состояний

Создание диаграммы состояний для класса «Заказ растрового клише» с указанием всех состояний и переходов между состояниями поможет в дальнейшем разработчикам окончательно понять, как надо писать код для этого класса.

Описание переходов:

1. На переходе от состояния «Инициализация» к состоянию «Выполнение заказа приостановлено» создают событие «Выполнить заказ».

2. Добавляют событие «Отменить заказ» к переходу между суперсостоянием и состоянием «Отменен».

3. На переходе от состояния «Выполнение заказа приостановлено» к состоянию «Выполнен» добавляют событие «Добавить к заказу новую позицию» с условием «Не осталось незаполненных позиций».

4. На рефлексивном переходе состояния «Выполнение заказа приостановлено» добавляют событие «Добавить к заказу новую позицию» с условием «Остаются незаполненные позиции».

11.3. Редактор VISIO и диаграмма состояний

VISIO как инструмент построения диаграмм состояний, пожалуй, несколько уступает Rational Rose, но среди трафаретов, появившихся после выполнения в VISIO команды Меню: File → New → Software → UML Model Diagram, имеется трафарет «UML Statechart», показанный на рис. 11.2.

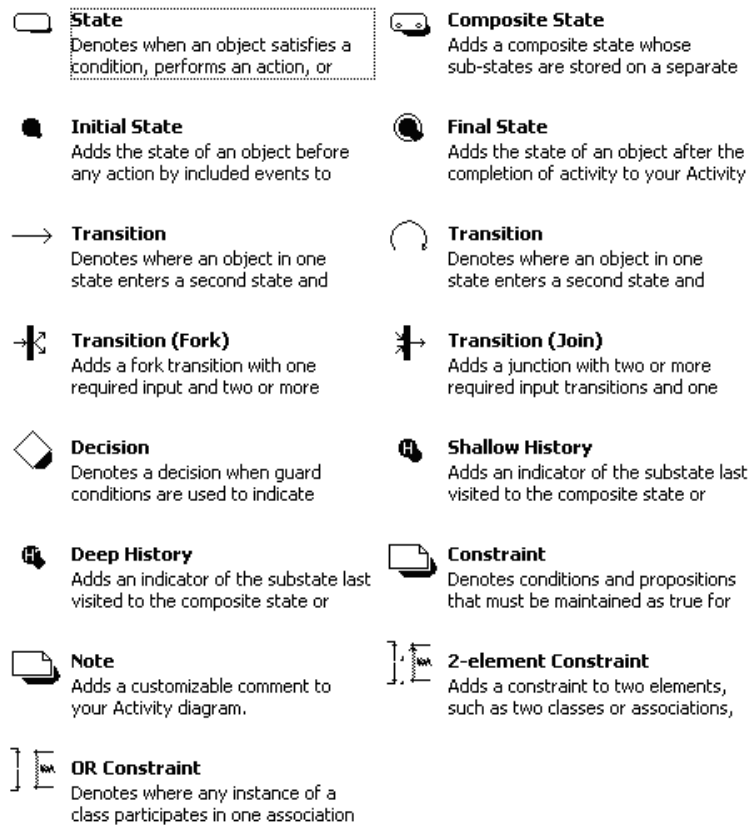


Рис. 11.2. Общий вид трафарета «UML Statechart»

Данный трафарет представляет собой шаблоны, необходимые для построения диаграмм состояний. Правда, на наш взгляд, работать с диаграммой состояний в Rational Rose удобней.

Вопросы для самоконтроля

1. Для чего предназначена диаграмма состояний Statechart?
2. Каковы основные элементы диаграммы Statechart?
3. Как создать диаграмму Statechart в Rational Rose?
4. Расскажите, как создать диаграмму Statechart в VISIO.
5. Какие другие инструменты для создания диаграммы Statechart вы знаете?

12. ДИАГРАММА ДЕЯТЕЛЬНОСТИ И ЕЕ РАЗРАБОТКА


12.1. Элементы нотации диаграммы деятельности и особенности ее построения

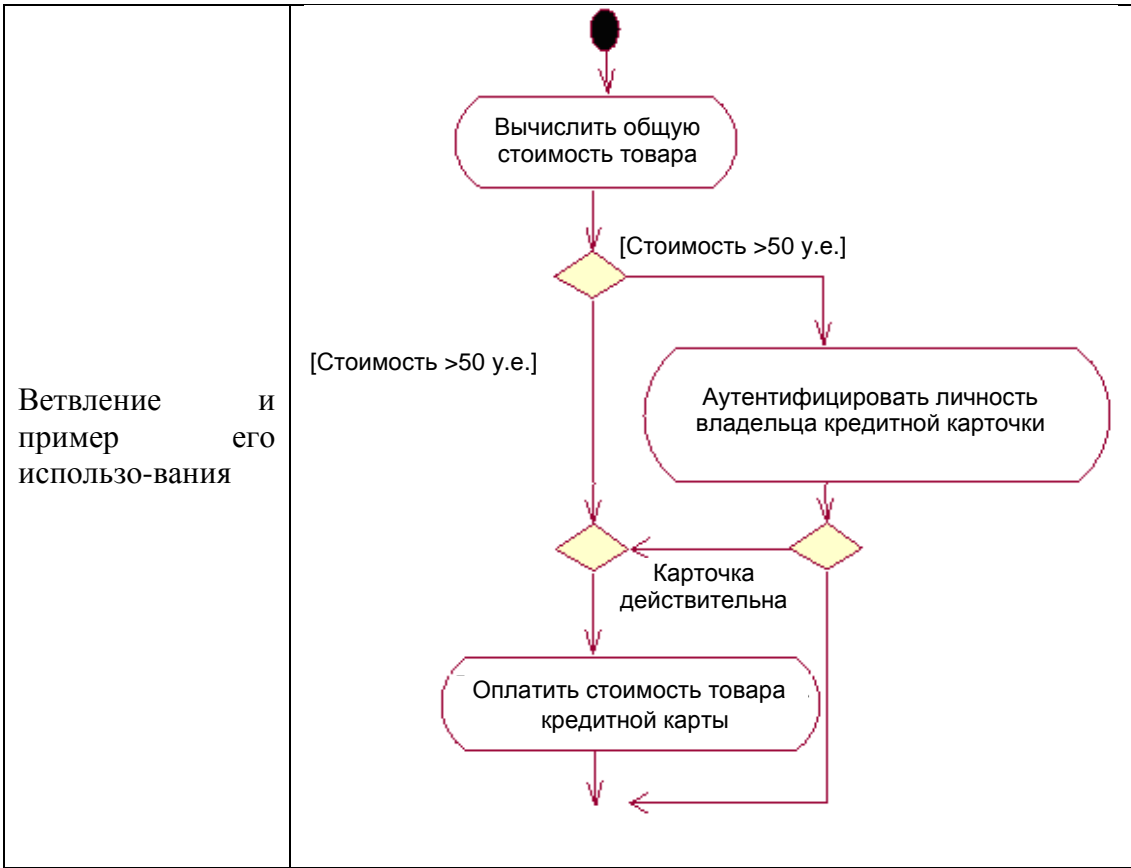
Диаграмма деятельности является альтернативой диаграмме состояний. Главные отличия между диаграммой деятельности и диаграммой состояний в том, что в первом случае основное – действие, а во втором – статическое состояние [10, 12–14].

Основные элементы и понятия, связанные с диаграммой деятельности, а также случаи ее использования представлены в табл. 12.1.

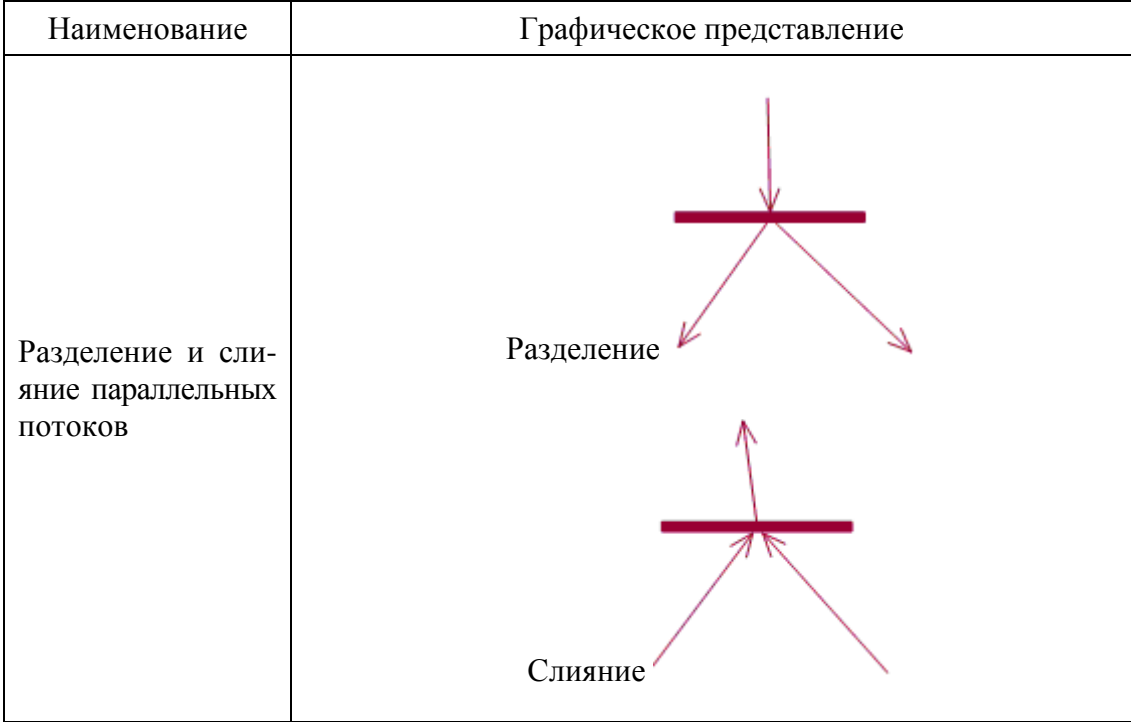
Таблица 12.1

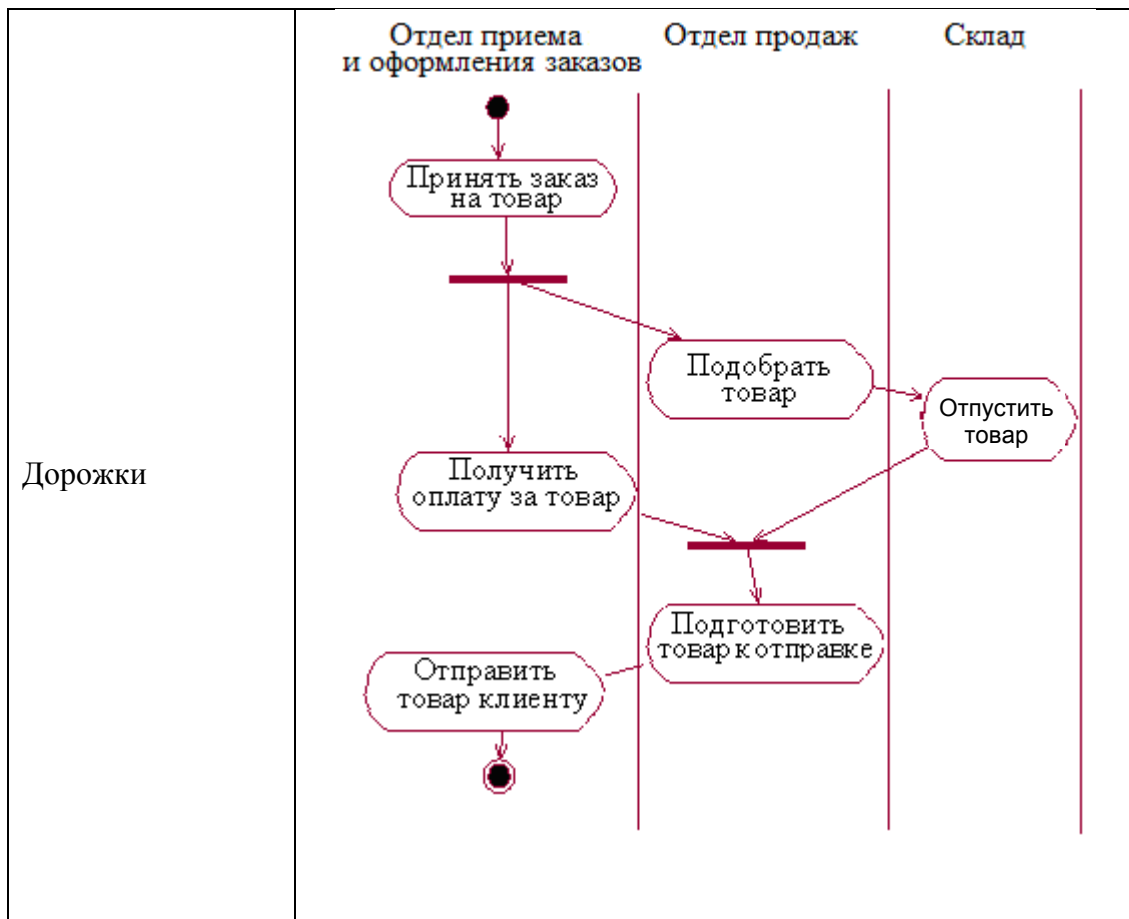
Основные элементы диаграммы деятельности

| Наименование | Графическое представление |
|--------------------|---|
| Состояние действия |  |



Окончание табл. 12.1





Состояние действия является специальным случаем состояния с некоторым входным действием и, по крайней мере, одним выходящим из состояния переходом.

Ветвление на диаграмме деятельности обозначается небольшим ромбом без текста внутри.

В приведенном выше примере рассчитывается общая стоимость товаров, покупаемых по кредитной карточке в супермаркете.

Разделение и слияние параллельных потоков позволяет проиллюстрировать ситуацию, когда параллельно выполняемые действия оказывают влияние друг на друга.

Дорожки могут быть использованы в случае, когда желательно выполнение каждого действия ассоциировать с конкретным подразделением компании.

12.2. Разработка диаграммы деятельности в Rational Rose

Упомянутые диаграммы представлены в табл. 12.2, где перечислены операции, связанные с построением диаграмм деятельности, и последовательность выполнения каждой из этих операций [15, 17, 19].

Таблица 12.2

Операции к построению диаграммы деятельности

| Операция | Рекомендуемая последовательность действий. Примечания |
|------------------------------------|--|
| 1. Создание диаграммы деятельности | Окно Browser: Use Case View → RClick → New → Activity Diagram → New Diagram ← = <Требуемое название> → DbClick – откроется окно диаграммы деятельности |
| 2. Создание действия | Панель Diagram: [Activity] → Окно диаграммы: LClick_^ – появится элемент действия, выбрав который, можно задать наименование действия |
| 3. Создание перехода | Панель Diagram: [State Transition] → Окно диаграммы: <Действие-источник> → LClick_∨ → <Действие-приемник> → LClick_^ |

Окончание табл. 12.2

| Операция | Рекомендуемая последовательность действий. Примечания |
|--------------------------------------|--|
| 4. Создание точки принятия решения | Панель Diagram: [Decision] → Окно диаграммы: LClick _^ ← = <Наименование точки принятия решения> → Панель Diagram: [State Transition] → Окно диаграммы: <Действие-источник> → LClick_∨ → <Точка принятия решения> → LClick_^ |
| 5. Создание контролируемого перехода | Панель Diagram: [State Transition] → Окно диаграммы: <Точка принятия решения> → LClick_∨ → <Действие-приемник> → <Линия перехода> → DbClick → State Transition Specification → Detail → |

| | |
|---|---|
| | поле Guard Condition = <Текст условия> → [OK] |
| 6. Приведение линии диаграммы к ортогональному виду | Окно диаграммы: <Преобразуемая линия> → LClick_^ → Меню: Format → Line Style → Rectilinear – при необходимости щелкнуть на линии и, не отпуская кнопку, переместить линию в нужном направлении |
| 7. Создание полосы синхронизации | Панель Diagram: {[Horizontal Synchronization] [Vertical Synchronization]} → Окно диаграммы: LClick_^ → Панель Diagram: [StateTransition] → <Снабдить полосу синхронизации необходимой входящей/исходящей связью> – повторить операцию для создания остальных связей-переходов |
| 8. Разделение диаграммы деятельности на зоны | Панель Diagram: [Swimlane] → Окно диаграммы: LClick_^ → NewSwimlane → DbClick → Swimlane Specification → Name = <Имя зоны> → [OK] – используя метод буксировки, можно перемещать границы зоны либо элементы диаграммы |
| 9. Создание исходного/завершающего действия | Панель Diagram: {[Start State] [End State]} → Окно диаграммы: LClick_^ → Панель Diagram: [State Transition] → Окно диаграммы: <Действие-источник> → LClick_∨ → <Действие-приемник> → LClick_^ |

Построение диаграммы деятельности при разработке ПО «Информационная система обработки полиграфических заказов».
Если представить предмет разработки как класс, то поведение программы, порожденное этим классом, может быть представлено в виде диаграммы деятельности, показанной на рис. 12.1.

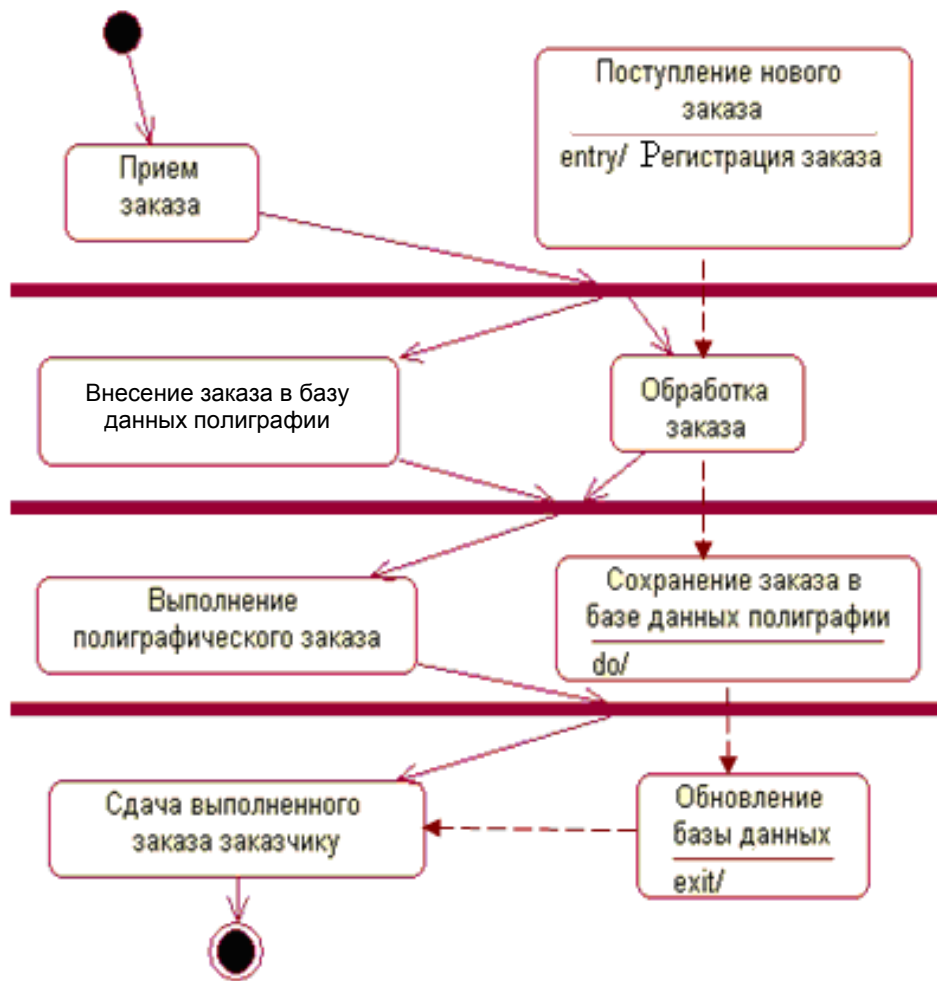


Рис. 12.1. Диаграмма деятельности

12.3. Редактор VISIO и диаграмма деятельности

VISIO как инструмент построения диаграмм деятельности, на наш взгляд, несколько уступает Rational Rose, но среди трафаретов, появившихся после выполнения в VISIO команды Меню: File → New → Software → UML Model Diagram, имеется трафарет «UML Activity», показанный на рис. 12.2. Он представляет шаблоны, необходимые для построения диаграмм деятельности.


















| | | | |
|---|---|---|--|
|  | Action State Adds an internal action and at least one related outgoing transition to |  | State Denotes when an object satisfies a condition, performs an action, or |
|  | Initial State Adds the state of an object before any action by included events to |  | Final State Adds the state of an object after the completion of activity to your Activity |
|  | Transition (Fork) Represents a Fork transition requiring one input and two or more |  | Transition (Join) Represents a Join transition requiring two or more input transitions and one |
|  | Control Flow Adds a transition from state one to another to your Activity diagram. |  | Object Flow Adds an object that shows flow to and from an action state to your |
|  | Object In State Adds an object that is manipulated by a number of successive activities |  | Decision Denotes a decision when guard conditions are used to indicate |
|  | Swimlane Defines a partition by assigning responsibility to action states in your |  | Signal Receipt Denotes where a signal call for an operation by an object is received in |
|  | Signal Send Denotes where a signal call for an operation by an object is sent in your |  | Constraint Denotes conditions and propositions that must be maintained as true for |
|  | Note Adds a customizable comment to your Activity diagram. |  | 2-element Constraint Adds a constraint to two elements, such as two classes or associations, |
|  | OR Constraint Denotes where any instance of a class participates in one association | | |

Рис. 12.2. Общий вид трафарета «UML Activity»

Вопросы для самоконтроля

1. Для чего предназначена диаграмма деятельности Activity?
2. Какие отличия между диаграммами Activity и Statechart?
3. Назовите основные элементы диаграммы Activity.
4. Как создать диаграмму Activity в Rational Rose?
5. Расскажите, как создать диаграмму Activity в VISIO.
6. Какие другие инструменты для создания диаграммы Activity вы знаете?

13. ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ И ЕЕ РАЗРАБОТКА

13.1. Элементы нотации диаграммы последовательности и особенности ее построения

Диаграмма последовательности используется для представления временных особенностей передачи и приема сообщений между объектами. Она позволяет наглядно показать сценарии вариантов использования разрабатываемого ПО [10–14].

Основные элементы и понятия, связанные с диаграммой последовательности, а также случаи ее использования представлены в табл. 13.1.

Объекты, непосредственно участвующие во взаимодействии, на диаграмме последовательности отображаются с линиями жизни и размещены последовательно слева направо, а неявное время представлено вертикалью, направленной сверху вниз.

Линия жизни служит для обозначения периода времени, в течение которого объект существует в системе. Отдельные объекты, выполнив свою роль в системе, могут быть уничтожены (разрушены).

Таблица 13.1

Основные элементы диаграммы последовательности

| Наименование | Графическое представление |
|-------------------|---------------------------|
| Основные элементы | |

| Наименование | Графическое представление |
|-------------------------|---------------------------|
| Дополнения к сообщениям | |

Фокус управления используется для представления периода активности объекта. Если активности объекта чередуются с периодами ожидания, то объект имеет несколько фокусов управления.

На ДП имеется возможность визуализации **рекурсии** в виде небольшого прямоугольника, присоединенного к правой стороне фокуса управления того объекта, для которого показывается это рекурсивное изображение.

Сообщение представляет собой законченный фрагмент информации, который отправляется одним объектом другому.

В отдельных случаях объект может пересылать сообщения самому себе, иницируя рефлексивные сообщения (показано в предыдущем примере). Каждое сообщение имеет направление от отправителя (клиента) к получателю (сервер).

В UML предусмотрено несколько разновидностей сообщений, каждое из которых имеет свое графическое изображение:

- 1) – вызов процедур, выполнение операций или обозначение отдельных вложенных потоков управления;
- 2) – простой поток управления асинхронного сообщения;
- 3) – асинхронное сообщение между двумя объектами в некоторой процедурной последовательности;
- 4) – возврат из вызова процедуры.

Для изображения ветвления рисуются две или более стрелки, выходящие из одной точки фокуса управления объекта.

В UML предусмотрены некоторые стандартные действия, выполняемые в ответ на получение соответствующего действия: Call (взять), Return (возврат), Create, Destroy, Send.

Send – посылка некоторому объекту некоторого сигнала, который асинхронно инициируется другим объектом. При этом сигнал должен быть описан в том классе, объект которого инициирует его передачу.

При необходимости на диаграммах последовательности можно задать временные ограничения в фигурных скобках ({}). Кроме того, в диаграммы последовательности можно включать комментарии.

13.2. Разработка диаграммы последовательности в Rational Rose

При построении диаграммы последовательности в Rational Rose аналитик может по своему усмотрению включать/отключать измерение сообщений, включать/выключать отображение фокусов управления. Операции к построению диаграммы последовательности и последовательность выполнения каждой из них приведены в табл. 13.2 [15, 17, 19].

Таблица 13.2

Операции к построению диаграммы последовательности

| Операция | Рекомендуемая последовательность действий. Примечания |
|--|--|
| 1. Создание диаграммы последовательности | Окно Browser: <Значок реализации прецедента> → RClick → New → Sequence Diagram → New Diagram ← = <Требуемое имя диаграммы последовательности> |
| 2. Создание объектов в диаграмме последовательности | Окно Browser: <Значок реализации прецедентов> → DbClick – откроется окно диаграммы → 1{Окно Browser: <Требуемый объект> → Click_∨ → Окно диаграммы: Click_∧ <Diagram> → [Object] → Окно диаграммы: Click → <New Object> ← = <Требуемое имя>}* – повторить для всех объектов сценария |
| 3. Создание сообщения в диаграмме последовательности | 1{Панель Diagram: [Object Message] → Окно диаграммы: <Объект-источник> → Click_∨ → <Объект-приемник> → Click_∧ → <Линия сообщения> → DbClick → Message Specification → Name = <Текст сообщения> → [OK]}* – повторить для всех сообщений |
| 4. Связывание объекта диаграммы | Окно Browser: <Требуемый класс> → Click_∨ → Окно диаграммы: <Требуемый объект> → Click_∧ |

| | |
|---------------------------------|--|
| последовательности с классом | |
|---------------------------------|--|

| Операция | Рекомендуемая последовательность действий. Примечания |
|-------------------------|---|
| 5. Связывание диаграмм | Панель Diagram: [Note] → Окно диаграммы: <Требуемая позиция> → Click → Окно Browser: <Элемент для ссылки> → Click_↵ → Окно диаграммы: <Объект-приемник> → Click_↵ |
| 6. Следование по ссылке | Окно диаграммы: <Требуемая ссылка-примечание> → DbClick |

Построение диаграммы последовательности при разработке ПО «Информационная система обработки полиграфических заказов». Ниже на рис. 13.1 представлен фрагмент сценария взаимодействия объектов, реализующих работу ПО «Информационная система обработки полиграфических заказов».

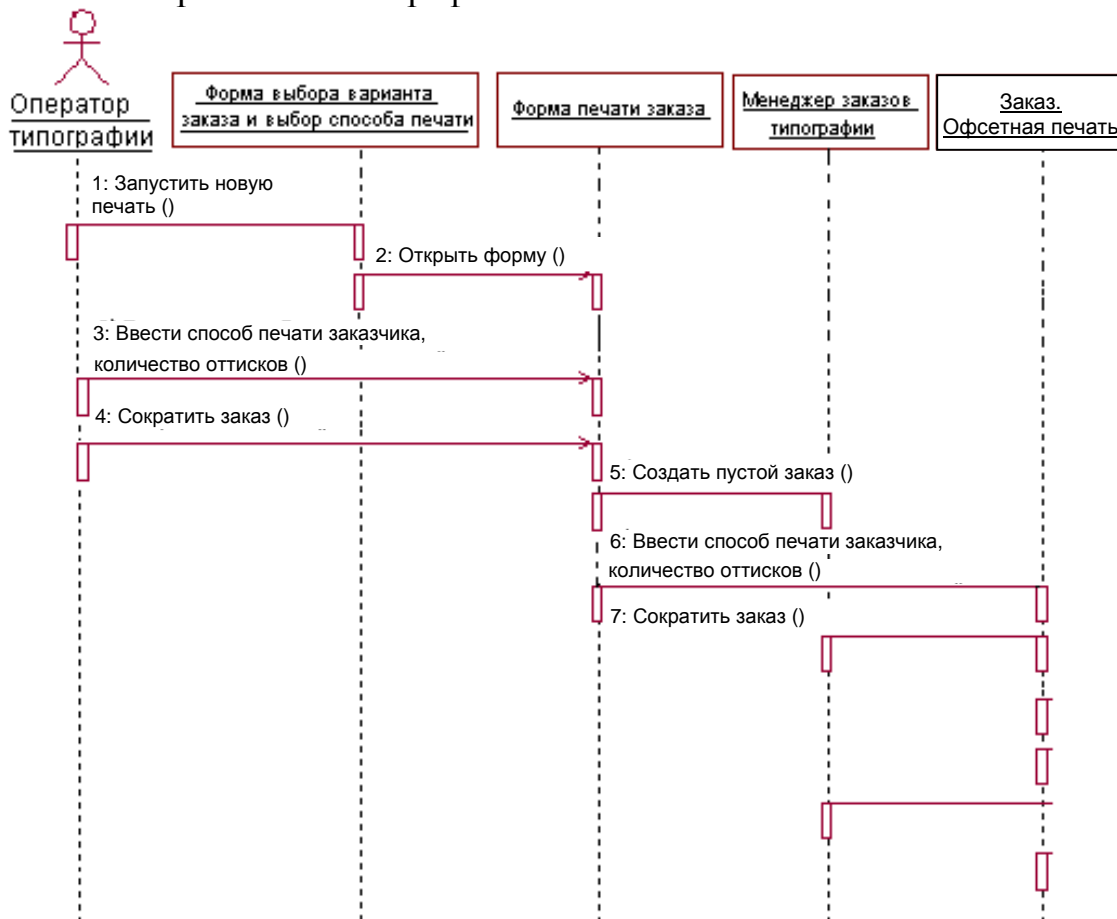


Рис. 13.1. Диаграмма последовательности

Эта диаграмма необходима для моделирования варианта использования «Ввести новый заказ». Она соответствует успешному варианту хода событий. Для описания того, что случится, если возникнет ошибка, или если пользователь выберет другие действия из предложенных, придется разработать другие диаграммы. Каждый альтернативный поток варианта использования может быть промоделирован с помощью своих собственных диаграмм взаимодействия.

13.3. Редактор VISIO и диаграмма последовательности

VISIO как инструмент построения диаграмм последовательности несколько уступает Rational Rose, но среди трафаретов, появившихся после выполнения в VISIO команды Меню: File → New → Software → UML Model Diagram, имеется трафарет «UML Sequence», показанный на рис. 13.2. Он представляет шаблоны для построения диаграмм последовательности.

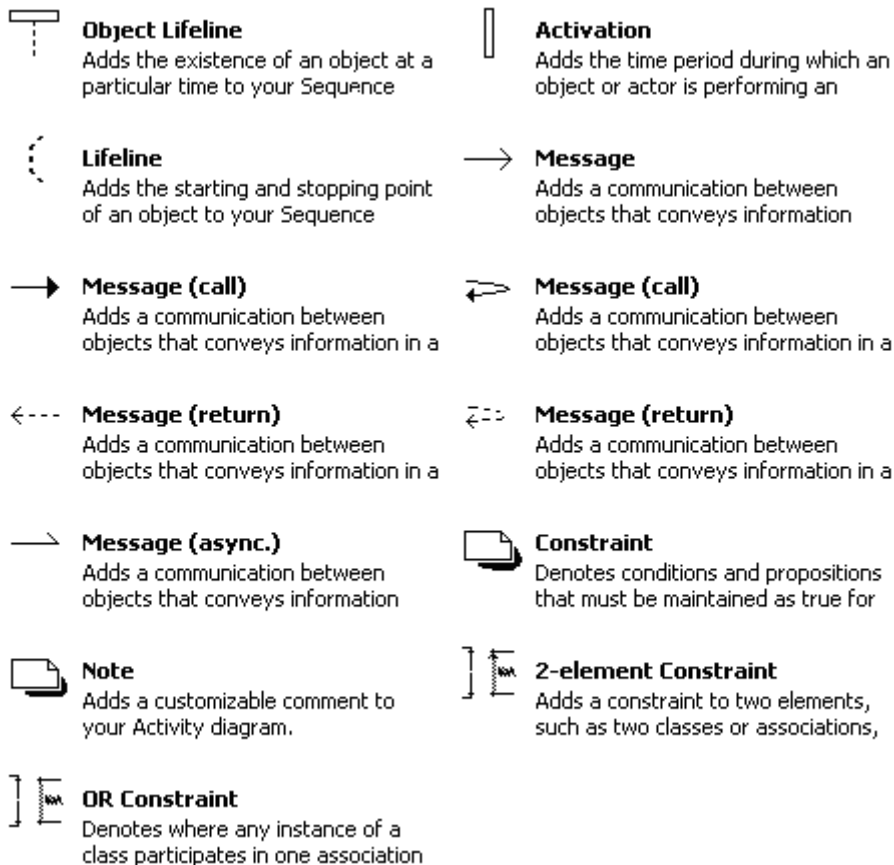


Рис. 13.2. Общий вид трафарета «UML Sequence»

Вопросы для самоконтроля

1. Для чего предназначена диаграмма последовательности Sequence?
2. Каковы основные элементы диаграммы Sequence?
3. Как создать диаграмму Sequence в Rational Rose?
4. Расскажите, как создать диаграмму Sequence в VISIO.
5. Какие другие инструменты для построения диаграммы Sequence вы знаете?
6. Что представляет собой сообщение и какие разновидности сообщений предусмотрены в UML?
7. Чем отличается линия жизни от фокуса управления?

14. ДИАГРАММА КООПЕРАЦИЙ И ЕЕ РАЗРАБОТКА

14.1. Элементы нотации диаграммы коопераций и особенности ее построения

Диаграмма коопераций является альтернативой диаграммы последовательности. Если диаграмма последовательности служит для визуализаций временных аспектов взаимодействия, то диаграмма коопераций предназначена для спецификации структурных аспектов взаимодействия [10, 12–14]. Следовательно, на диаграмме лучше просматривается распределение процессов между объектами (из-за отсутствия оси неявного времени).

Кооперация может быть представлена на двух уровнях:

– на уровне спецификации (показывает роли классов и ассоциации в рассматриваемом взаимодействии);

– на уровне примеров (определяются необходимые свойства объектов, участвующих в кооперации, а также указываются ассоциации, которые должны иметь место в кооперации).

Поскольку одна совокупность объектов может учитываться в различных кооперациях, на кооперации уровня спецификации указываются лишь те свойства операторов и связей, которые необходимы в данной кооперации, а на диаграмме классов должны быть все свойства и ассоциации между элементами диаграммы (рис. 14.1).



Рис. 14.1. Кооперация уровня спецификации

Отметим, что подобное представление используется на начальных этапах проектирования. В последующем каждая из коопераций подлежит детализации на уровне примеров. При этом в качестве

элементов диаграммы коопераций выступают объекты и связи, дополненные сообщениями.

Основные элементы и понятия, связанные с диаграммой коопераций уровня примеров, а также случаи ее использования представлены в табл. 14.1.

Объекты являются основными графическими примитивами, из которых строится диаграмма коопераций на уровне примеров. При этом возможные варианты записи строки текста в прямоугольнике объекта могут быть следующими:

- 1) :C – анонимный объект, образуемый на основе класса C;
- 2) /R – анонимный объект, играющий роль R;
- 3) /R:C – анонимный объект, образуемый на основе класса C и играющий роль R;
- 4) O/R – объект O, играющий роль R;
- 5) O:C – объект O, образуемый на основе класса C;
- 6) O/R:C – объект O, образуемый на основе класса C и играющий роль R;
- 7) O – объект O;
- 8) /R – роль с именем R;
- 9) :C – анонимная роль на базе класса C (не подчеркнута);
- 10) /R:C – роль R на основе класса C.

Таблица 14.1

Основные элементы диаграммы кооперации уровня примеров

| № п/п | Наименование | Графическое представление |
|-------|--------------|---|
| 1 | Объекты | <p> <u>Клиент/инициатор запроса</u> <i>a</i> <u>/Инициатор запроса</u> <i>б</i> <u>/Обработчик запросов:</u> <u>Сервер</u> <i>в</i> <u>Клиент/инициатор запроса</u> <i>г</i> <u>/Обработчик запросов</u> <i>д</i> <u>:Клиент::База данных</u> <i>е</i> </p> |
| 2 | Мультиобъект | <p>Мультиобъект</p> |

| | | |
|---|-----------------|---|
| 3 | Активный объект | <div style="border: 1px solid black; padding: 5px; display: inline-block;"> a: Вызывающий абонент </div> 1:создать → <div style="border: 1px solid black; padding: 5px; display: inline-block;"> c:Соединение </div> |
|---|-----------------|---|

Окончание табл. 14.1

| № п/п | Наименование | Графическое представление |
|-------|------------------|---------------------------|
| 4 | Составной объект | |
| 5 | Связи | |

Выше в табл. 14.1 даны примеры представления объектов на диаграммах кооперации.

Здесь в п. 1 табл. 14.1 (e) и (d) иллюстрируют классы применительно к уровню спецификации. Причем в (d) указана роль анонимного класса. В (e) имя класса уточнено именем пакета.

В UML объекты делятся на активные и пассивные. **Пассивный объект** оперирует только данными и не может инициировать деятельность по управлению другими объектами. Однако они могут посылать сигналы в процессе выполнения получаемых запросов.

Активный объект имеет свою собственную нить управления и может инициировать деятельность по управлению другими объектами. На диаграммах такие объекты отображаются прямоугольником с более широкими границами.

Составной объект, или объект-контейнер, предназначен для представления объекта, имеющего собственную структуру и внутренние потоки управления.

На диаграммах коопераций составной объект отображается как обычно, а в нижней секции вместо атрибутов указываются составные части этого объекта.

Связь является экземпляром или примером произвольной ассоциации. Бинарная связь на диаграмме коопераций изображается отрезком прямой, соединяющим два объекта.

На каждом из концов может быть указаны имена ролей данной ассоциации. Связи не имеют собственных имен.

Связь может иметь некоторые стереотипы, записываемые рядом с одним из ее концов. Стереотипы указывают на особенность реализации данной связи.

В UML могут использоваться следующие стереотипы:

- «*association*» – ассоциация. Предполагается по умолчанию;
- «*parameter*» – параметр метода (составной объект может быть только параметром некоторого метода);
- «*local*» – локальная переменная метода (видима только в пределах соседнего объекта);
- «*global*» – глобальная переменная (ее видимость распространяется на всю диаграмму коопераций);
- «*self*» – рекурсивная связь объекта с самим собой.

К сказанному ранее по части сообщений при рассмотрении диаграммы последовательности добавим, что в диаграмме коопераций каждое сообщение может быть помечено строкой текста следующего формата:

<Предшествующие сообщения> <[Сторожевое условие]>
<Выражение последовательности> <Возвращаемое значение:= имя сообщения> <Список аргументов>

Ниже приведены примеры записи:

– *предшествующих сообщений*:

A3, B4/C5: ошибка записи (сектор);

A3, B4 – сообщение не может быть передано, пока не будут переданы своим адресатам сообщения A3, B4;

- сторожевого условия:
[количество цифр номера равно 7] 2.1: набрать_тел_номер();
- выражения последовательности:
[количество цифр номера равно 7] 3.1: набрать_тел_номер();
- возвращаемого значения:
1.2.3: P:= ...;
- имени сообщения:
1.2.3: P:= найти_документ(...);
- списка аргументов:
1.2.3: P:= найти_документ (спецификация документа).

14.2. Разработка диаграммы коопераций в Rational Rose

В Rational Rose пожалуй, самый простой способ построения диаграммы коопераций. Это подтверждается приведенной ниже табл. 14.2 [15, 17, 19].

Таблица 14.2

Преобразование диаграммы последовательности в диаграмму коопераций

| Операция | Рекомендуемая последовательность действий. Примечания |
|--|--|
| Создание диаграммы коопераций на основе диаграммы последовательности | Окно Browser: <Элемент, соответствующий диаграммам последовательностей> → DbClick → {Меню: Browse → Create Collaboration Diagram [F5]} |

Построение диаграммы коопераций при разработке ПО «Информационная система обработки полиграфических заказов». Альтернативой к рассмотренному в п. 13.2 сценарию может быть диаграмма коопераций, представленная на рис. 14.2. На ней лучше просматриваются аспекты взаимодействия объектов, но хуже сценарий из-за отсутствия неявной оси времени.

Напомним, что для построения диаграммы коопераций по диаграмме последовательности при использовании средства Rational Rose достаточно нажать функциональную клавишу F5.

При детальном рассмотрении диаграммы коопераций можно увидеть, что для ее построения используются следующие классы (объекты) и сообщения между ними.

Сообщения – запустить новую печать; открыть форму; ввести способ печати, заказчика, количество оттисков; сохранить заказ; создать пустой заказ; передача для оформления заказа.



Рис. 14.2. Collaboration Diagram

Классы (объекты) – форма выбора варианта заказа; выбор способа печати; форма печати заказа; заказ офсетная печать; позиция печатного заказа; менеджер транзакций; менеджер заказов типографии.

14.3. Редактор VISIO и диаграмма коопераций

VISIO как инструмент построения диаграмм коопераций существенно уступает Rational Rose, но среди трафаретов, появившихся после выполнения в VISIO команды Меню: File → New → Software → UML Model Diagram, имеется трафарет «UML Collaboration», показанный на рис. 14.3. Он представляет шаблоны для построения диаграмм коопераций.

Поскольку диаграммы взаимодействия (Sequence и Collaboration), в основном, применяются для моделирования сценариев

вариантов использования, представляющих пользовательский интерфейс, то имеет смысл взаимодействие объектов дополнять макетами экранных форм пользовательского интерфейса. В этом плане VISIO может оказать неоценимые услуги. Так, при выборе в VISIO команды Меню: File → New → Software → Windows User Interface появляются к услугам пользователя трафареты, показанные на рис. 14.4. Они представляют собой ряд шаблонов для построения макета пользовательского интерфейса.

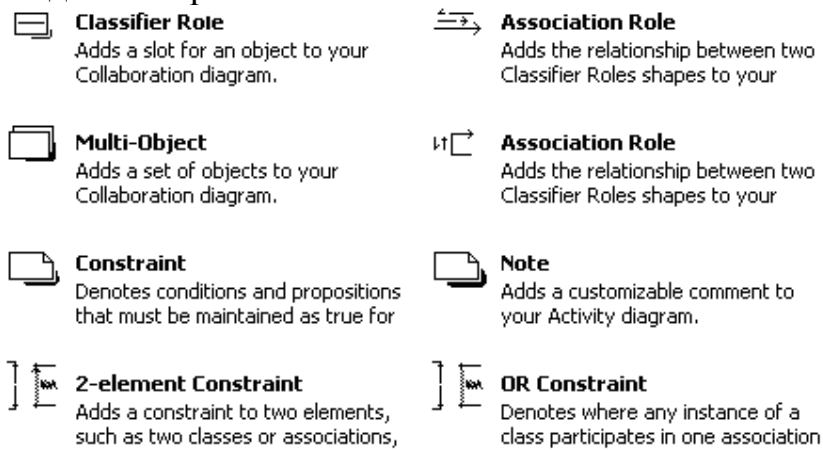


Рис. 14.3. Общий вид трафарета «UML Collaboration»

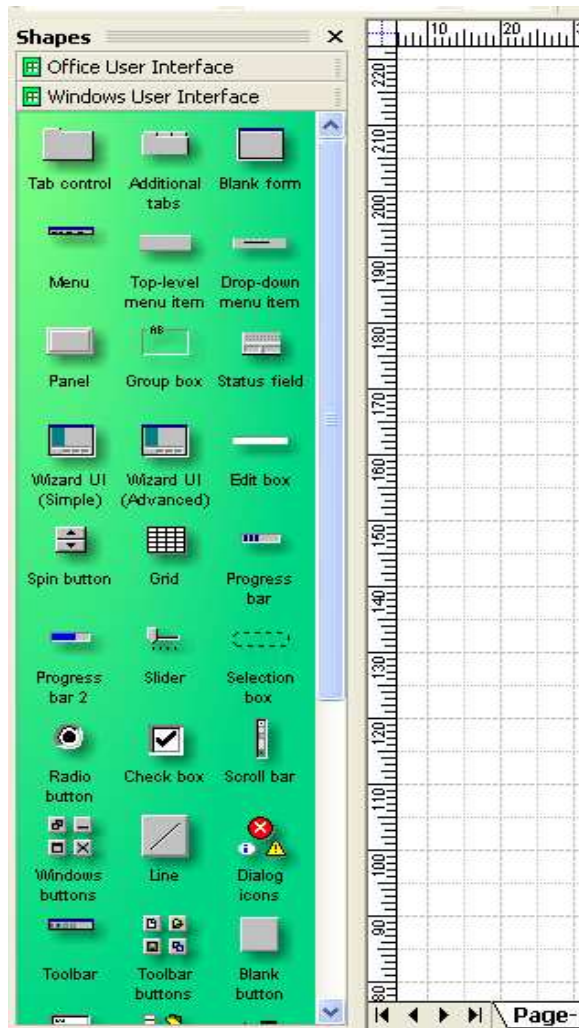


Рис. 14.4. Фрагмент окна с трафаретом «Office User Interface» и «Windows User Interface»

Вопросы для самоконтроля

1. Для чего предназначена диаграмма коопераций Collaboration?
2. Каковы отличия между диаграммами Collaboration и Sequence?
3. Расскажите, как создать диаграмму Collaboration в Rational Rose.
4. Как создать диаграмму Collaboration в VISIO?
5. Какие другие инструменты для построения диаграммы Collaboration вы знаете?
6. Какие стереотипы предусмотрены в UML для указания особенности связи?
7. Чем отличается обозначение активного объекта от пассивного?

15. ДИАГРАММА КЛАССОВ И ЕЕ РАЗРАБОТКА

15.1. Элементы нотации диаграммы классов и особенности ее построения




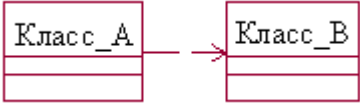
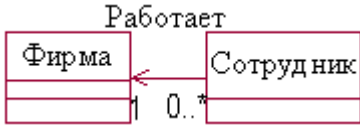
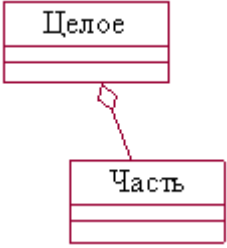
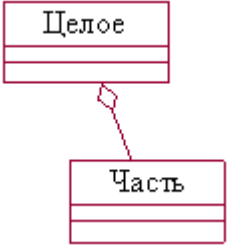
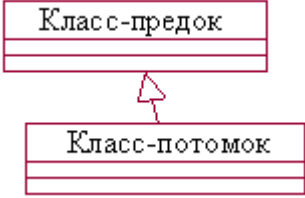
Диаграммы классов служат для представления статической структуры модели системы в терминологии объектно-ориентированного подхода [10, 12–14].

Основные элементы и понятия, связанные с диаграммами классов, представлены в табл. 15.1.

Таблица 15.1

Основные элементы диаграммы классов

| Наименование | Графическое представление |
|--|---------------------------|
| Односекционный значок класса | |
| Двухсекционный значок класса | |
| Трёхсекционный значок класса | |
| Четырёхсекционный значок класса | |
| Специализированные классы | |
| Параметризованный класс (Parametrized class) | |

| Наименование | Графическое представление |
|---------------------------------------|---|
| Граничный класс (Boundary class) |  Класс_А |
| Управляющий класс (Control class) |  Класс_А |
| Класс-сущность (Entity class) |  Класс_А |
| Отношения между классами | |
| Отношение зависимости |  |
| Отношение ассоциации |  |
| Отношение агрегации |  |
| Отношение композиции |  |
| Отношение обобщения (наследование) |  |

Класс (Class) служит для обозначения множества объектов с одинаковой структурой, поведением и отношениями с объектами других классов и может быть в UML представлен следующими значками.

Односекционный значок класса содержит необязательный стереотип класса, обязательное имя класса.

Двухсекционный значок класса дополнительно к предыдущему содержит атрибуты класса, каждый из которых может быть представлен строкой вида:

<Квантор видимости> <Имя атрибута> [Кратность]: <Тип атрибута> = <Исходное значение> {Строка-свойство}.

Трёхсекционный значок класса дополнительно к предыдущему содержит операции класса, каждая из которых может быть представлена строкой вида:

<Квантор видимости> <Имя операции> (Список параметров): <Тип возвращаемого значения> {Строка-свойство}.

Четырёхсекционный значок класса дополнительно к предыдущему содержит исключения.

Параметризованный класс (Parametrized class) предназначен для обозначения такого класса, который имеет один или более нефиксированных формальных параметров.

Граничный класс (Boundary class) порождает объекты, с которыми взаимодействует актер.

Управляющий класс (Control class) порождает объекты, управляющие взаимодействием объектов.

Класс-сущность (Entity class) порождает объекты для долговременного хранения информации.

Отношение зависимости имеет место тогда, когда некоторое изменение источника зависимости (например, класс_В) может потребовать изменения другого, зависящего от него, клиента зависимости (например, класс_А).

Отношение ассоциации имеет место при наличии некоторого отношения между классами – классом «Фирма» и классом «Сотрудник».

Отношение агрегации представляет системные взаимосвязи типа «часть – целое».

| | |
|--|---|
| 7. Создание основной диаграммы классов | Окно диаграммы: <Требуемый пакет> → DblClick – откроется окно диаграммы классов пакета → 1{Окно Browser: <Требуемый класс пакета> → Click_↵ → Окно диаграммы классов пакета: LClick_^}* |
|--|---|

Продолжение табл. 15.2

| Операция | Рекомендуемая последовательность действий. Примечания |
|--|---|
| 8. Установка признака отображения принадлежности класса к пакету | Меню: Tools → Options → <Вкл. Diagram> → <Группа Companiments> → [Show Visibility] → [OK] → Окно диаграммы: <Требуемый класс> → RClick → <Options> → [Show Visibility] |
| 9. Создание связи ассоциации классов | Панель Diagram: [Association] → Окно диаграммы: <Символ 1-го класса> → Click_↵ → <Символ 2-го класса> → LClick_^ |
| 10. Создание связи агрегирования классов | Панель Diagram: [Aggregation] → Окно диаграммы: <Класс – «целое»> → Click_↵ → <Класс – «часть»> → LClick_^ |
| 11. Именованье связи классов | Окно диаграммы: <Требуемая связь> → DblClick → <Диалог Association Specification> → Name → <Имя связи> → [OK] |
| 12. Именованье роли связи классов | Окно диаграммы: <Конец связи> → RClick → Role Name → <Имя роли> |
| 13. Заданье значения признака множественности | Окно диаграммы: <Линия связи> → DblClick → Association Specification → {Role A Detail Role B Detail} → Multiplicity = <Требуемое значение> → [OK] |
| 14. Создание возвратной связи | Панель Diagram: {[Association] [Aggregation]} → Окно диаграммы: <Символ требуемого класса> → LClick_↵ → <Конец первой части связи> → LClick_^ ← LClick_↵ → <Конец второй части связи> → LClick_^ → <Обозначить имена полей и признаки множественности для обоих концов связи> |
| 15. Создание связи пакетов | Панель Diagram: {[Dependency] _____} → Окно диаграммы: <Пакет-клиент> → Click_↵ → <Пакет-сервер> → Click_^ |
| 16. Соотнесение сообщения на диаграмму взаимодействия с операцией класса | Диаграмма взаимодействия: <Линии сообщения> → RClick → <New operation> → <Operation Specification> → <Вкл. General> → Name = {<Имя операции новое> <Имя операции из списка>} |

| | |
|--------------------------------------|--|
| 17. Создание операции класса | Окно Browser: <Требуемый класс> → RClick → New → Operation → Orname ← = <Требуемое имя операции> |
| 18. Документирование операции класса | Окно Browser: <Значок класса> → [+] → <Элемент требуемой операции> → Окно Documentation: = <Текстовое описание операции> |

Продолжение табл. 15.2

| Операция | Рекомендуемая последовательность действий. Примечания |
|---|--|
| 19. Создание атрибута класса | Окно Browser: <Значок класса> → RClick → New → Attribute → Name ← = <Требуемое имя атрибута> |
| 20. Документирование атрибута | Окно Browser: <Значок класса> → [+] → <Элемент требуемого атрибута> → Окно Documentation: = <Текстовое описание атрибута> |
| 21. Создание диаграммы классов пакета | Окно Browser: <Значок класса> → RClick → New → Class Diagram → New Diagram ← = <Требуемое имя диаграммы классов> |
| 22. Включение классов в диаграмму классов | Окно Browser: <Значок требуемой диаграммы класса> → DbClick – откроется окно диаграммы → Меню: Query → Add Classes → <Список Package> → <Нужный пакет> → <Список Classes> → <Выделить нужные классы> → [>>>>] → [OK] |
| 23. Фильтрация связей на диаграмме классов | Окно Browser: <Значок требуемой диаграммы> → DbClick → Меню: Query → Filter Relationship → <Диалог Relations> → <Группа Type> → [Name] → [OK] |
| 24. Отображение некоторых атрибутов и/или операций на диаграмме классов | Окно диаграммы: <Требуемый класс> → RClick → Options → Select Compartment Items → <Диалог Edit component> → <Список All Items> → <Выделить требуемые элементы> → [>>>>] → [OK] |
| 25. Отображение всех атрибутов на диаграмме классов | Окно диаграммы: <Требуемый класс> → RClick → Options → Show All Attributes |
| 26. Отображение всех операций на диаграмме классов | Окно диаграммы: <Требуемый класс> → RClick → Options → Show All Operations |
| 27. Установка режима отображения атрибутов (операций) по | Меню: Tools → RClick → Options → <Диалог Options> → {[Show All Attributes] [Show All Operations]} → [OK] |

| | |
|---|--|
| умолчанию | |
| 28. Задание режима отображения стереотипа класса по диаграмме классов | Окно диаграммы: <Значок требуемого класса> → RClick → Options → Stereotype Display → {None Label Icon Decoration} |
| 29. Создание класса ассоциаций | <Открыть окно диаграммы> → <Разместить класс ассоциаций> → Панель Diagram: [Association Class] → <Значок класса-ассоциаций> → Click_↵ → <Ассоциация тарификации> → Click_^ |

Продолжение табл. 15.2

| Операция | Рекомендуемая последовательность действий. Примечания |
|--|---|
| 30. Создание диаграммы иерархии наследования классов | <Открыть окно диаграммы> → <Разместить на диаграмме требуемые классы> → 1{<Панель Diagram> → [Generatization] → <Символ подкласса> → Click_↵ → <Символ суперкласса> → Click_^}* |
| 31. Перемещение атрибутов по иерархии наследования классов | Окно Browser: 1{<Требуемый подкласс> → [+] → <Элемент атрибута> → Click_↵ → <Суперкласс> → Click_^ → <Удалить атрибут из остальных подклассов иерархии>}* |
| 32. Перемещение операций по иерархии наследования классов | Окно Browser: 1 {<Требуемый подкласс> → [+] → <Элемент операции> → Click_↵ → <Суперкласс> → Click_^ → <Удалить операцию из остальных подклассов иерархии>}* |
| 33. Придание пакету статуса глобального | Окно диаграммы: <Требуемый пакет> → RClick → Open Specification → Package Specification → Detail → <Флажок Global> → [OK] |
| 34. Создание диаграммы классов для реализации варианта использования | Окно Browser: <Элемент реализации необходимого прецедента> → RClick → New → Class Diagram → New Diagram ← = <Требуемое имя диаграммы> → DbClick → 1{<Элемент требуемого класса> → Click_↵ → <Окно диаграммы> → Click_^}* – для всех классов, включаемых в диаграмму |
| 35. Задание направления связи | Окно диаграммы: <Требуемый конец связи> → RClick → Navigation |
| 36. Задание принадлежности связанных классов | Окно диаграммы: <Необходимая связь агрегирования> → DbClick → Aggregation Specification → {Role A Detail Role B Detail} → Containment → {[By Value] [By Reference]} → [OK] |

| | |
|---|--|
| 37. Создание связи зависимости | Панель Diagram: [Dependency] → Окно диаграммы: <Класс-клиент> → Click_∨ → <Класс-сервер> → Click_^ |
| 38. Задание типа и исходного значения атрибута класса | Окно Browser: <Символ необходимого класса> → RClick → Open Specification → Class Specification → Attributes → <Требуемый атрибут в списке> → RClick → Specification → <Class Attribute Specification> → Type = <Выбранный из списка или набранный тип> → (Initial Value = <Исходное значение>) → [OK] → [OK] |

Окончание табл. 15.2

| Операция | Рекомендуемая последовательность действий. Примечания |
|---|--|
| 39. Задание сигнатуры операции класса | Окно Browser: <Необходимый класс> → RClick → Open Specification → Class Specification → Operations → <Требуемая операция из списка> → RClick → Specification → Operation Specification → General → Return Type = <Выбранный из списка или набранный тип> → Detail → Arguments → RClick → Insert → Argname → Name = <Имя> → Type = <Тип> → Detail = <Значение по умолчанию> → [OK] → [OK] |
| 40. Задание режима отображения сигнатур операций классов на диаграмме | Меню: Tools → Options → Diagram → [Show Operation Signature] |
| 41. Изменение режима отображения сигнатур операций отдельного класса | Окно диаграммы: <Требуемый класс> → RClick → Options → Show Operation Signature |

В случае ассоциации классов с языком реализации при построении диаграммы классов раскрывается не только вся прелесть сервиса, предоставляемого этим режимом, но и преимущества работы в Rational Rose, интегрированном с такими средами разработки, как Visual Studio, VC++, VB и др. При этом программист может в интерактивном режиме переходить с модели приложения к ее исходному коду и наоборот, редактируя то и другое.

Построение диаграммы классов при разработке ПО «Информационная система обработки полиграфических заказов».

Классы (объекты)– форма выбора варианта заказа; выбор способа печати; форма печати заказа; заказ офсетная печать; позиция печатного заказа; менеджер транзакций; менеджер заказов типографии.

Все классы группируются в пакеты.

Пакеты – Границы, Сущности, Управление (рис. 15.1).

Классы:

- для пакета «Сущности» – заказ растрового клише; позиция заказа;
- для пакета «Границы» – форма печати; выбор способа печати;
- для пакета «Управление» – менеджер транзакций; менеджер заказов типографии.

Создание диаграммы классов для сценария «Ввести новый заказ» со всеми классами. Диаграммы классов для вышеупомянутого сценария, рассредоточенные по типам классов (классы-сущности, управляющие классы, граничные классы), представлены на рис. 15.2–15.4.



Рис. 15.1. Главная диаграмма классов обработки заказов

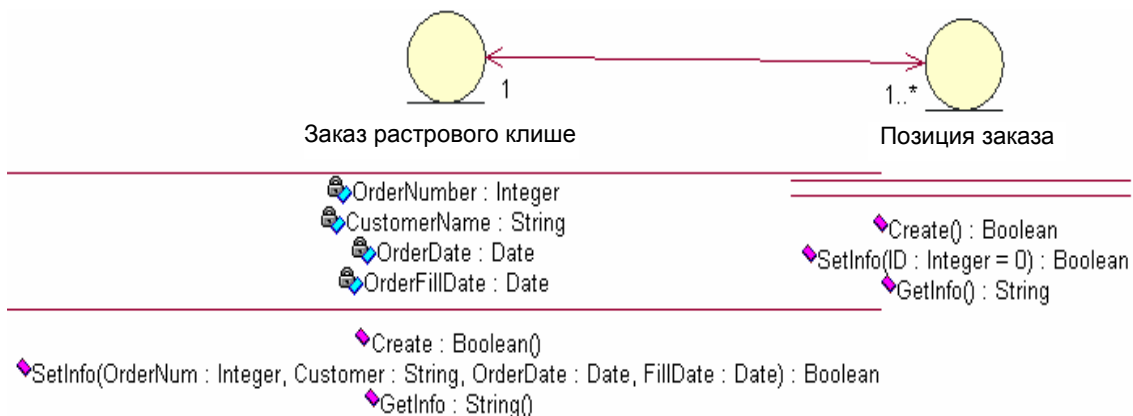


Рис. 15.2. Диаграмма классов пакета «Сущности»

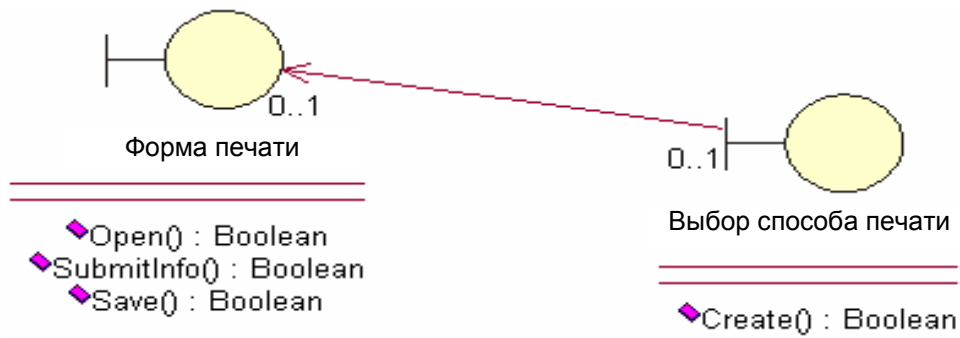


Рис. 15.3. Диаграмма классов пакета «Границы»

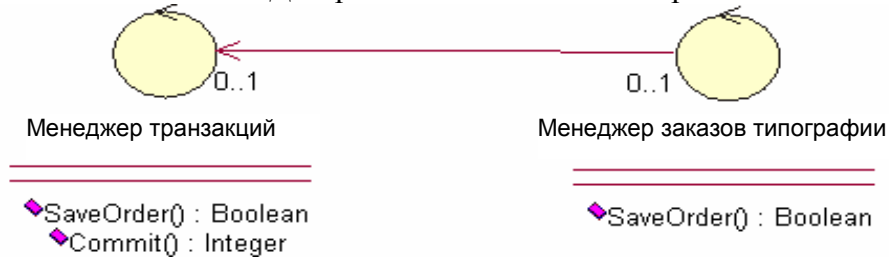


Рис. 15.4. Диаграмма классов пакета «Управление»

Добавление атрибутов и операций.

Добавляют атрибуты и операции к классам диаграммы классов «Ввести новый заказ». Для атрибутов и операций используют специфические для языка особенности. Устанавливают параметры так, чтобы показывать все атрибуты, все операции и их сигнатуры. Видимость показывают с помощью нотации UML (рис. 15.5).

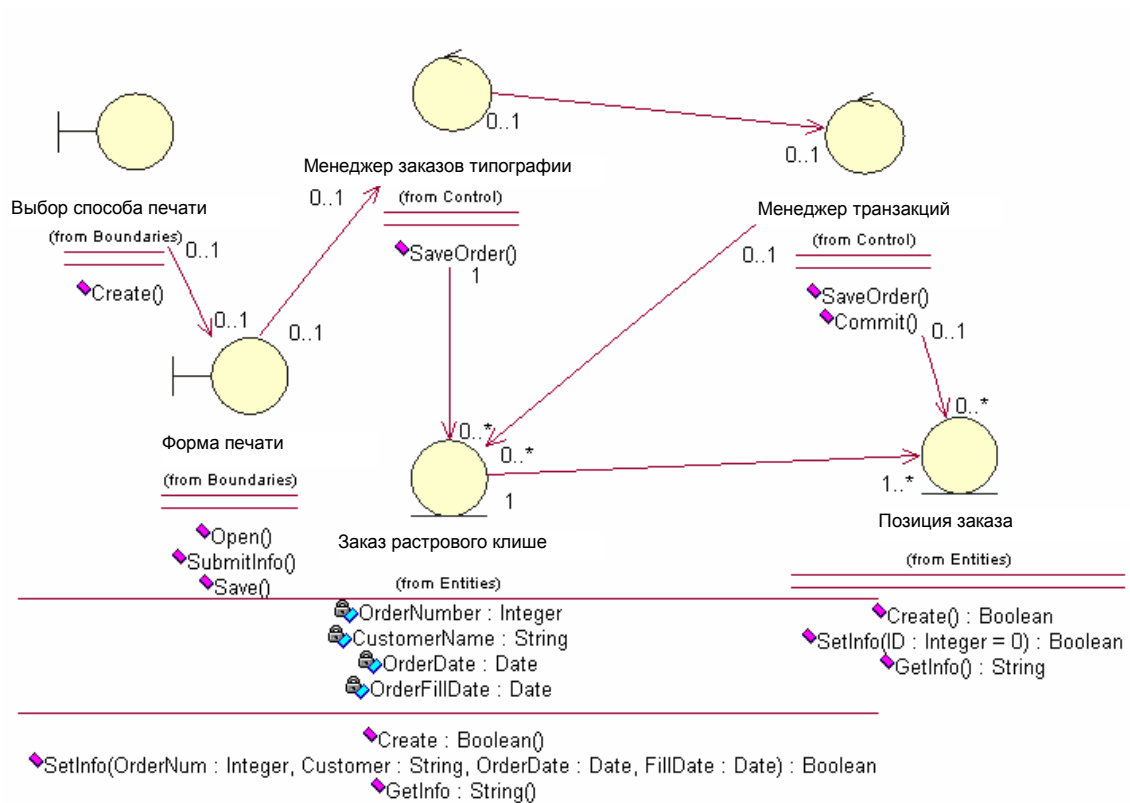


Рис. 15.5. Ассоциации сценария «Ввести новый заказ»

Добавление ассоциаций.

1. От класса «Выбор способа печати» к классу «Форма печати».
2. От класса «Форма печати» к классу «Менеджер заказов типографии».

| | | | |
|--|--|--|---|
|  Package Adds a grouping of model elements to your Component diagram. |  Class Denotes a set of objects that have similar structure, behavior, and |  Data Type Adds a data type to your Static Structure diagram. |  Interface Adds the externally visible operations of a class, component, package, or |
|  Interface Adds the externally visible operations of a class, component, package, or |  Generalization Adds a classification relationship between a more general and a more |  Binary Association Adds a customizable relationship between exactly two classes to your |  Composition Adds a collaboration in which all of the participants are part of one |
|  Association Class Adds attributes, operations, and other properties to an association in |  Dependency Denotes where there is a semantic relationship between two model |  Utility Adds the grouping of global variables and procedures in the form of a class |  Subsystem Adds a subsystem to your Static Structure diagram. |
|  Parameterized Class Adds the descriptor for a class that has one or more unbound formal |  Binding Connects (binds) designated arguments to a template's formal |  Bound Element Adds the result of a binding between a template's parameters and actual |  Object Adds a particular instance of a class to your Deployment diagram. |
|  Link Adds a link between two objects to your Static Structure diagram. |  N-ary Link Adds a link between three or more objects to your Static Structure |  Metaclass Adds a class whose instances are classes to your Static Structure |  Signal Adds a signal to your Static Structure diagram. |
|  Exception Adds an exception signal to your Static Structure diagram. |  Trace Adds a semantic relationship between two model elements from |  Refinement Adds the fuller specification of a model element that has already been |  Usage Adds a dependency relationship where one element requires another |
|  Note Adds a customizable comment to your Activity diagram. |  Constraint Denotes conditions and propositions that must be maintained as true for |  2-element Constraint Adds a constraint to two elements, such as two classes or associations, |  OR Constraint Denotes where any instance of a class participates in one association |
|  N-ary Association Adds an association among three or more classifiers to your Static |  N-ary AssocClass Adds an association between three or more classifiers that also has class | | |

Рис. 15.6. Общий вид трафарета «UML Static Structure»

3. От класса «Менеджер заказов типографии» к классу «Заказ растрового клише», к классу «Менеджер транзакций».

4. От класса «Заказ растрового клише» к классу «Позиция заказа».

5. От класса «Менеджер транзакций» к классу «Позиция заказа», к классу «Заказ растрового клише».

15.3. Редактор VISIO и диаграмма классов

VISIO как инструмент построения диаграмм классов менее гибок, чем Rational Rose, однако среди трафаретов, появившихся после выполнения команды Меню: File → New → Software → UML Model Diagram, имеется трафарет «UML Static Structure», показанный на рис. 15.6. Он представляет шаблоны для построения диаграммы классов.

Вопросы для самоконтроля

1. Каково назначение диаграммы классов?
2. Назовите основные элементы диаграммы классов.
3. Какие виды связей доступны в диаграмме классов?
4. Для чего используется каждый вид связи?
5. Как создать диаграмму классов в Rational Rose?
6. Расскажите, как создать диаграмму классов в VISIO.
7. Какие другие инструменты для построения диаграммы классов вы знаете?

16. ДИАГРАММА КОМПОНЕНТОВ И ЕЕ РАЗРАБОТКА

16.1. Элементы нотации диаграммы компонентов и особенности ее построения

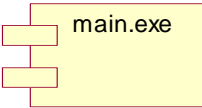
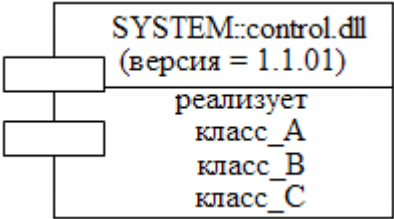
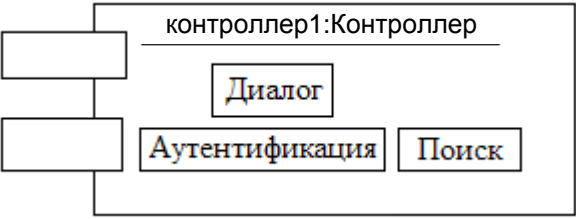
Диаграмма компонентов описывает особенности физического представления системы [10, 12–14]. В ее состав входят компоненты, интерфейсы, отношения, примечания и ограничения. Диаграммы компонентов обычно используются:

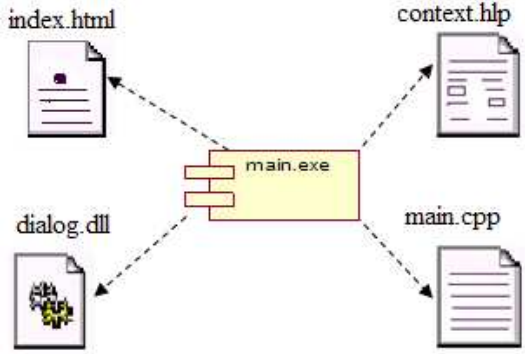
- для моделирования исходного кода;
- моделирования исполняемой версии;
- моделирования физических БД;
- моделирования адаптивных систем.

Основные элементы и понятия, связанные с диаграммами компонентов, представлены в табл. 16.1.

Таблица 16.1

Основные элементы диаграммы компонентов

| Наименование | Графическое представление |
|-------------------------------------|--|
| Односекционный значок компонента |  |
| Двухсекционный значок компонента |  |
| Значок компонента уровня экземпляра |  |
| Специальные значки компонентов |  |

| Наименование | Графическое представление |
|-----------------------|--|
| Отношение зависимости |  |

Компонент – это физическая сущность, которая реализует некоторый набор интерфейсов и может быть в UML представлена следующими значками.

Односекционный значок компонента содержит имя компонента, а также при необходимости имя пакета и помеченное значение.

Двухсекционный значок компонента может содержать дополнительную информацию, например имя пакета и помеченное значение. В нижней же секции компонента можно указать информацию о реализуемых им классах.

Значок компонента уровня экземпляра содержит подчеркнутое имя компонента, написанное с малой буквы. В нижней секции значка при этом могут быть указаны объекты, реализуемые объектом.

В UML выделяют 3 вида компонентов:

- компоненты развертывания (LIBRARY, TABLE);
- компоненты «рабочие продукты» (FILE, DOCUMENT);
- компоненты исполнения (exe-файлы).

Специальные значки компонентов часто используют для упрощения понимания диаграммы. Этим подчеркивают привязку реализации компонентов к конкретной технологии. На каноническом изображении для этой цели указывают явно стереотип компонента (LIBRARY, TABLE, FILE и т. д.) перед его именем.

Зависимость служит для представления факта наличия такой связи, когда изменение одного элемента модели оказывает влияние или приводит к изменению другого элемента модели.

Отношение зависимости на диаграмме компонента изображается пунктирной линией, направленной от зависимого элемента к независимому.

16.2. Разработка диаграммы компонентов в Rational Rose

Особенности разработки диаграммы компонентов представлены в табл. 16.2, где перечислены операции, связанные с их построением, и последовательность выполнения каждой из этих операций [15, 17, 19].

Таблица 16.2

Операции к построению диаграммы компонентов

| Операция | Рекомендуемая последовательность действий. Примечания |
|---|---|
| 1. Создание пакета уровня реализации | Окно Browser: <Component View> → RClick → New → Package → New Package ← = <Требуемое имя пакета> |
| 2. Создание основной диаграммы компонентов | Окно Browser: <Component View > → <Main> → DbClick → 1{<Требуемый пакет> → Click_√ → Окно диаграммы: Click_^}* – для всех пакетов, включаемых в диаграмму → 1{Панель Diagram: [Depen-dency] → Окно диаграммы: <Пакет-клиент> → Click_√ → <Пакет-сервер> → Click_^}* – для всех связей зависимости диаграммы |
| 3. Установка соответствия между классом и компонентом | Окно Browser: <Требуемый компонент> → RClick → Open Specification → Component Specification → Releases → <Требуемый класс> → RClick → Assign → [OK] |

Построение диаграммы компонентов при разработке ПО «Информационная система обработки полиграфических заказов». Создаются диаграммы компонентов «Границы», «Сущности», «Управление» с зависимостями (рис. 16.1–16.3, 16.5).

Разработка диаграммы компонентов предполагает использование информации не только о логическом представлении модели системы, но и об особенностях ее физической реализации. В первую очередь, необходимо решить, из каких физических частей или файлов будет состоять программная система. На этом этапе следует обратить внимание на такую реализацию системы, которая обеспечивала бы возможность повторного использования кода за счет рациональной декомпозиции компонентов, а также создание объектов только при их необходимости.

Диаграммы компонентов для вышеупомянутого сценария, рассредоточенные по типам классов (классы-сущности, управляющие классы, граничные классы), представлены на рис. 16.2, 16.3, 16.5. Каждая из диаграмм включает в себя тело пакета и его спецификацию.

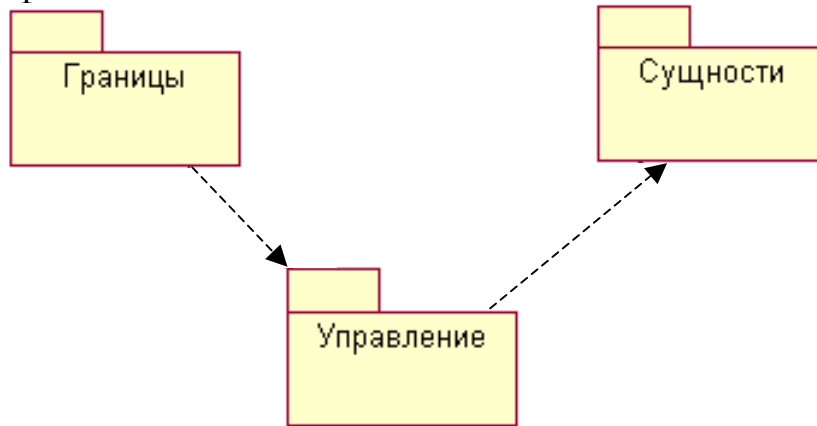


Рис. 16.1. Главная диаграмма компонентов системы

Светлыми значками (рис. 16.2–16.5) представлены спецификации компонентов, соответствующие в С++ файлам с расширением `.h`, а затемненными – тела компонентов, соответствующие в С++ файлам с расширением `.cpp`. Нотация Г. Буча.

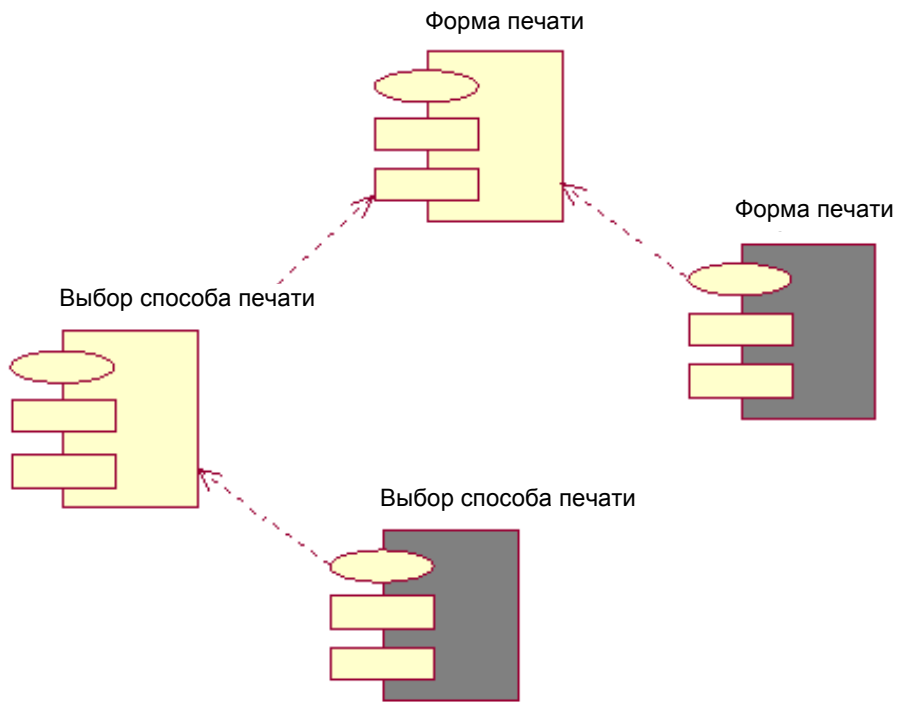


Рис. 16.2. Диаграмма компонентов пакета «Границы»

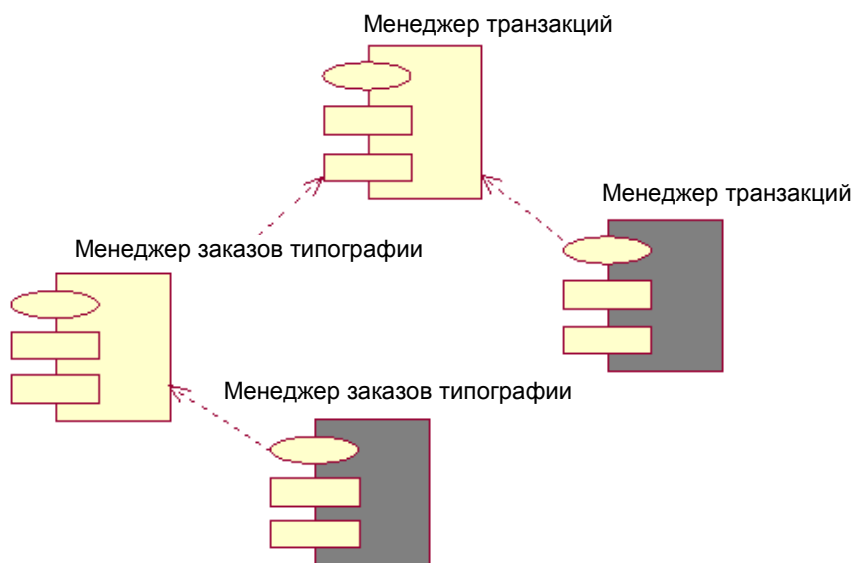


Рис. 16.3. Диаграмма компонентов пакета «Управление»

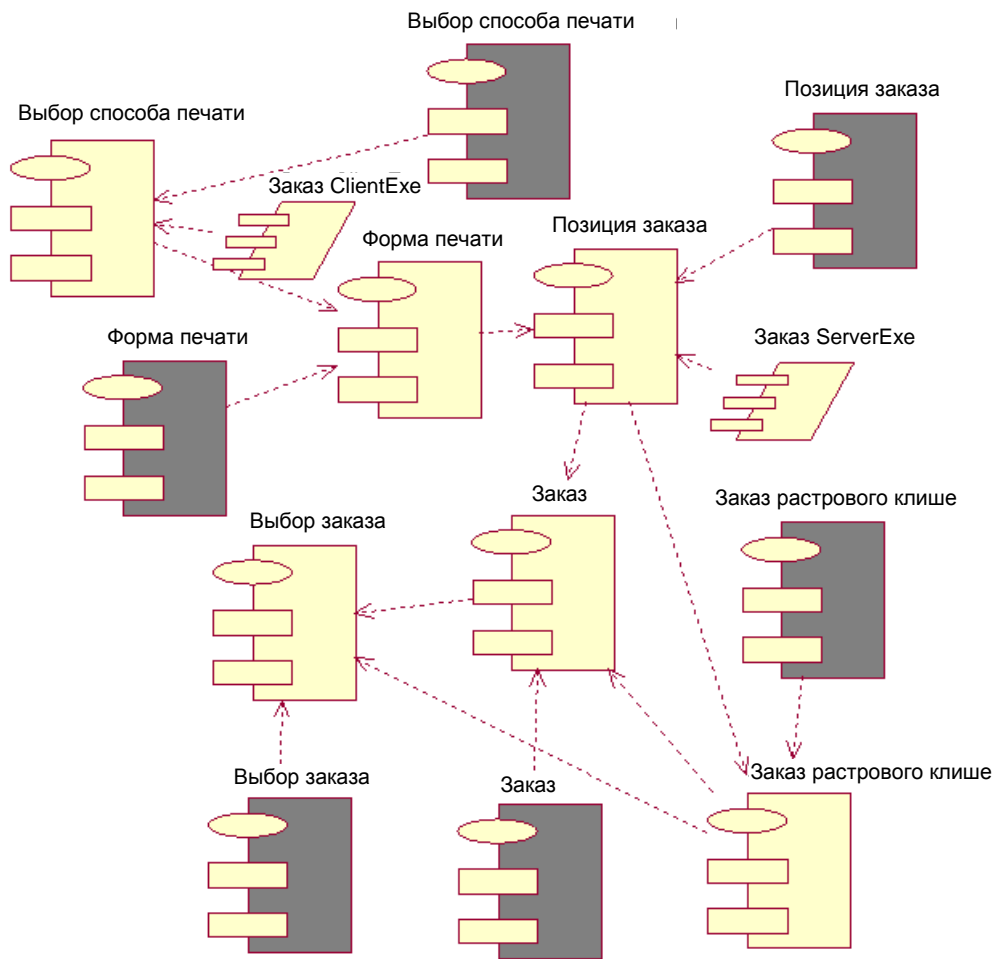


Рис. 16.4. Диаграмма компонентов системы

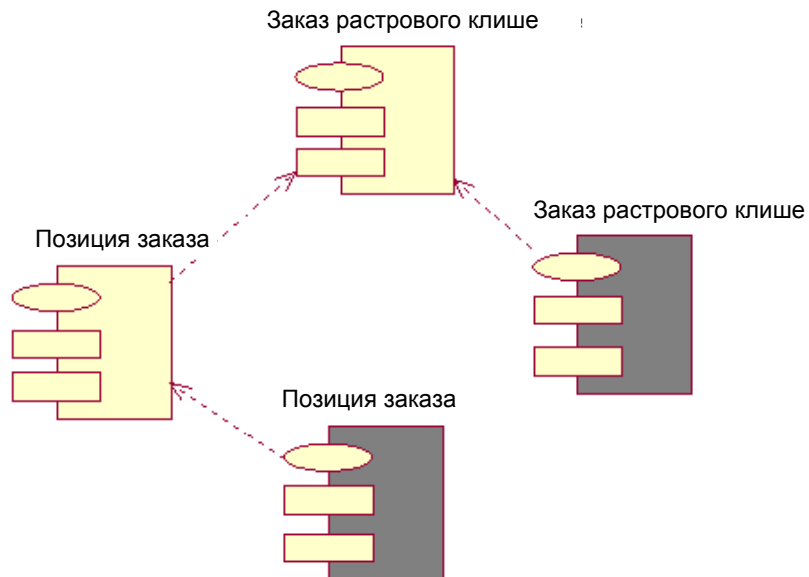


Рис. 16.5. Диаграмма компонентов пакета «Сущности»

16.3. Редактор VISIO и диаграмма компонентов

VISIO как инструмент построения диаграмм компонентов, пожалуй, не уступает Rational Rose. Среди трафаретов, появившихся после выполнения команды Меню: File → New → Software → UML Model Diagram, имеется трафарет «UML Component», показанный на рис. 16.6. Он представляет шаблоны для построения диаграмм компонентов.

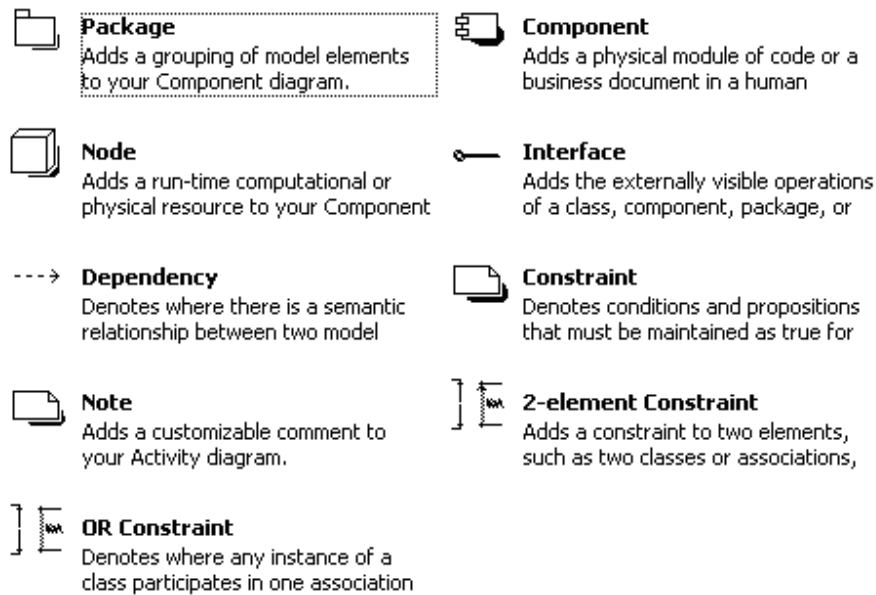


Рис. 16.6. Общий вид трафарета «UML Component»

Вопросы для самоконтроля

1. Для чего предназначена диаграмма компонентов Component?
2. Каковы основные элементы диаграммы компонентов?
3. Как создать диаграмму компонентов в Rational Rose?
4. Расскажите, как создать диаграмму компонентов в VISIO.
5. Какие другие инструменты для построения диаграммы компонентов вы знаете?
6. К какому компоненту направлена стрелка отношения зависимости?
7. Для чего используются специальные значки компонентов вместо канонических?

17. ДИАГРАММА РАЗВЕРТЫВАНИЯ И ЕЕ РАЗРАБОТКА

17.1. Элементы нотации диаграммы развертывания и особенности ее построения

Диаграмма развертывания применяется для представления общей конфигурации и топологии распределенной программной системы и содержит распределение компонентов по отдельным ее узлам [10, 12–14]. Диаграмма развертывания позволяет:

- определить распределение компонентов системы по ее физическим узлам;
- показать физические связи между всеми узлами реализации системы на этапе ее исполнения;
- выявить узкие места системы и реконфигурировать ее топологию для достижения требуемой производительности.

Диаграмма развертывания обычно разрабатывается совместно с системными аналитиками, сетевыми инженерами и системотехниками.

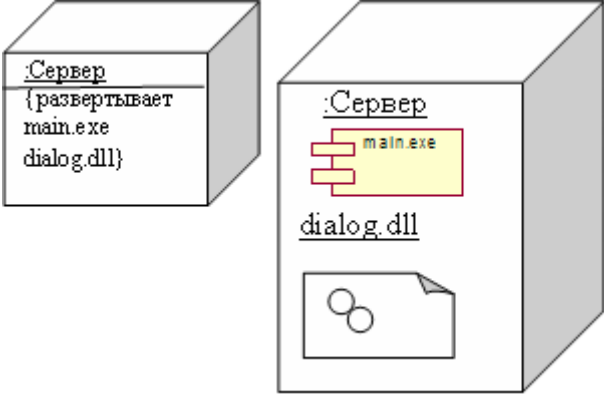
Основные элементы и понятия, связанные с диаграммами развертывания, представлены в табл. 17.1.

Узел представляет собой некоторый физически существующий элемент системы, обладающий некоторым вычислительным ресурсом, и может быть в UML представлен следующими значками.

Таблица 17.1

Основные элементы диаграммы развертывания

| Наименование | Графическое представление |
|---------------------------|---------------------------|
| Узел-тип и узел-экземпляр | |
| Узел со спецификой | |

| Наименование | Графическое представление |
|---------------------|--|
| Узел с компонентами |  |

На диаграмме развертывания узлы могут представляться как в качестве типов, так и в качестве экземпляров.

В первом случае имя узла записывается без подчеркивания и начинается с заглавной буквы. Во втором – имя узла-экземпляра записывается в следующем виде: <имя узла: имя типа узла>.

Изображения узлов могут содержать дополнительную информацию о специфике узла. Если эта информация относится к имени узла, то она записывается под этим именем в форме помеченного значения.

Указать явно компоненты, размещаемые на отдельном узле, можно двумя способами: с использованием списка компонентов либо их изображений.

Важно помнить, что в качестве таких вложенных компонентов могут выступать только исполняемые компоненты.

В литературе в качестве дополнения к имени узла можно встретить стереотипы, такие как «процессор», «датчик», «модем», «сеть», «консоль» и т. д. Сами же узлы могут быть изображены не только параллелепипедами, но и несколько иначе, например, процессор можно изобразить и в форме внешнего вида компьютера, что успешно можно реализовать с помощью шаблонов VISIO.

Соединения являются разновидностью ассоциации и изображаются отрезками линий без стрелок. Эти линии указывают на необходимость организации физических каналов для обмена информацией между узлами. Характер соединения может быть дополнительно специфицирован примечанием, помеченным значением и ограничением.

17.2. Разработка диаграммы развертывания в Rational Rose

Операции, связанные с построением диаграмм развертывания (размещения), и последовательность выполнения каждой из этих операций приведены в табл. 17.2 [15, 17, 19].

Таблица 17.2

Операции к построению диаграммы развертывания

| Операция | Рекомендуемая последовательность действий. Примечания |
|---|---|
| 1. Создание диаграммы развертывания | Окно Browser: Deployment View → DbClick → {Панель Diagram: [Processor] → Окно диаграммы: <Появившийся элемент> = <Наименование>} – для всех узлов диаграммы |
| 2. Введение связей между узлами диаграммы | Панель Diagram: [Connection] → <Первый узел> → Click_∨ → <Второй узел> → Click_∧ |

Следует отметить, что в части построения диаграмм развертывания Rational Rose уступает по возможностям графическому редактору VISIO. Последний позволяет совместить диаграмму развертывания с диаграммой компонентов. К тому же значки узлов можно показать более оригинально, что допускается синтаксисом UML.

Построение диаграммы развертывания при разработке ПО «Информационная система обработки полиграфических заказов». Возможность включения персонала в понятие узла не рассматривается в нотации языка UML, тем не менее, подобное расширение понятия узла позволяет создавать средствами языка UML модели самых различных систем, включая бизнес-процессы и технические комплексы. Действительно, для реализации бизнес-процессов компаний удобно рассматривать в качестве узлов-ресурсов системы организационные подразделения, состоящие из персонала. Автоматизация управления полиграфическим комплексом может потребовать рассмотрения в качестве самостоятельного элемента человека-оператора, способного принимать решения в нестандартных ситуациях и нести ответственность за возможные последствия этих решений.

Наиболее известны два специальных графических стереотипа для обозначения разновидностей узлов. Первый обозначает ресурсоемкий узел (processor), под которым понимается узел с процессором и памятью, необходимыми для выполнения исполняемых компонентов. Он изображается в форме куба с боковыми гранями, окрашенными в серый цвет. Второй стереотип в форме обычного куба обозначает устройство (device), под которым понимается узел без процессора и памяти. На этом типе узлов не могут размещаться исполняемые компоненты программной системы (рис. 17.1).

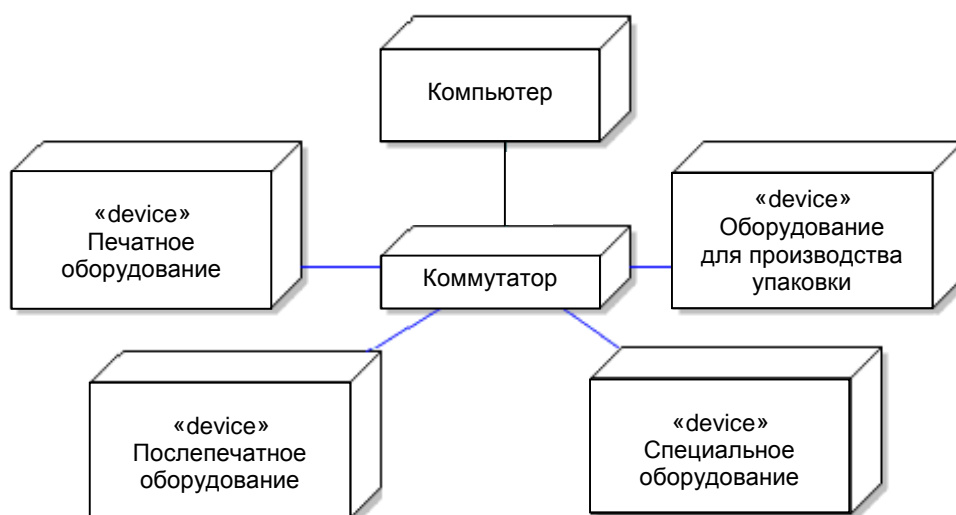


Рис. 17.1. Deployment Diagram

Для рассматриваемой полиграфической системы будем иметь следующие устройства и процессоры:

- 1) процессоры – коммутатор, компьютер;
- 2) устройства – печатное оборудование, послепечатное оборудование, специальное оборудование, оборудование для производства упаковки.

17.3. Редактор VISIO и диаграмма развертывания

VISIO как инструмент построения диаграмм развертывания (размещения) превосходит Rational Rose. Дело в том, что трафарет «UML Deployment», который появляется в группе трафаретов после выполнения команды Меню: File → New → Software → UML Model Diagram, позволяет создавать диаграммы развертывания с

изображением на узлах компонентов. Упомянутый трафарет показан на рис. 17.2.

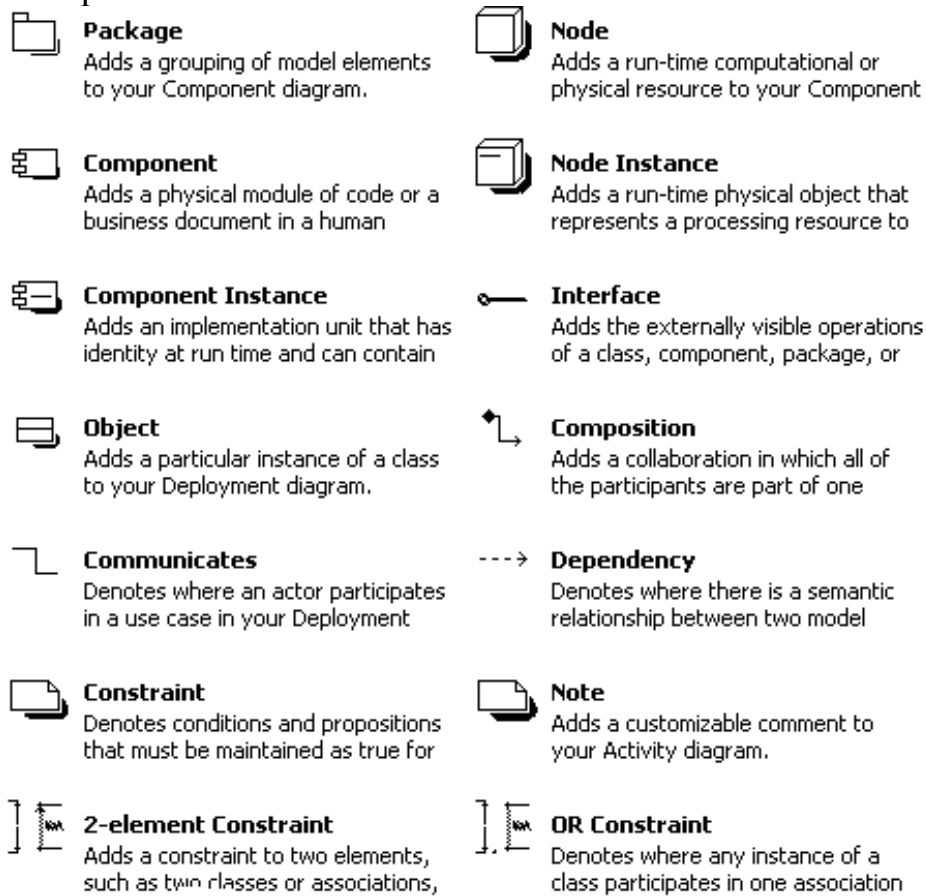


Рис. 17.2. Общий вид трафарета «UML Deployment»

Вопросы для самоконтроля

1. Для чего используется диаграмма развертывания Deployment?
2. Каковы основные элементы диаграммы Deployment?
3. Как создать диаграмму развертывания в Rational Rose?
4. Расскажите, как создать диаграмму развертывания в VISIO.
5. Какие другие инструменты для построения диаграммы развертывания вы знаете?

18. КОДОГЕНЕРАЦИЯ

18.1. Понятие генерации исходных текстов программ

Как правило, разработчики сталкиваются с необходимостью создания больших классов. При ручном вводе и объявлении имеется ряд подводных камней: во-первых, постановщик задачи обычно описывает «что нужно» на словах, в крайнем случае с минимальным бумажным сопровождением; во-вторых, разработчик системы зачастую игнорирует все комментарии, которыми необходимо сопровождать программный код.

При разработке больших систем коллективом программистов, когда код программы создает ряд исполнителей, имеется опасность нарушить общий замысел аналитика. Вследствие чего части кода трудно стыкуются. Генерация каркаса программы по модели исключает такие неприятные случаи [15–17, 19]. Когда каркас программы готов, заполнение его кодом можно поручить многим исполнителям, не опасаясь нарушить целостность проекта.

18.2. Генерация исходного кода в Rational Rose

Процесс генерации кода в Rational Rose состоит из четырех шагов:

1. Проверка корректности модели.
2. Установка свойств генерации кода.
3. Выбор класса, пакета или компонента.
4. Генерация кода.

Для проверки корректности модели следует выбрать в Меню: Tools → Check Model и проанализировать все найденные ошибки в окне журнала.

Установка и анализ свойств генерации кода производится в окне Tools → Options на вкладке соответствующего класса.

За один раз можно сгенерировать код для класса, пакета, компонента, а также для всех элементов соответствующей диаграммы. Выбор элемента (класса, пакета, компонента) для генерации кода производится на диаграмме или в браузере.

В процессе генерации кода (Меню: Tools → <Язык> → Code Generation) Rational Rose выбирает информацию из логического и компонентного представлений модели и генерирует «скелетный» код с учетом установленного набора свойств (или принятого по умолчанию).

Просмотреть результат генерации кода (тексты исходных файлов) можно через Меню: Tools, а сообщения об ошибках, возникающих в процессе генерации кода, – в окне журнала.

Rational Rose позволяет не только генерировать код по модели, но и восстанавливать модель по коду.

Операции, выполняемые при генерации кода C++, и рекомендуемая последовательность их реализации приведена в табл. 18.1, а при обратном восстановлении модели по коду C++ – в табл. 18.2.

Таблица 18.1

Операции к генерации кода C++

| Операция | Рекомендуемая последовательность действий. Примечания |
|---|--|
| 1. Создание набора свойств | Меню: Tools → Options → C++ → {Type = <Требуемый тип свойств из списка> → [Clone] → Clone Properties Set = <Наименование нового набора свойств> → [OK] → 1{<Model Properties> → <Выбрать свойства и щелкнуть в пределах столбца Value> → 1{Ввести новое значение свойств Выбрать из списка}}* – повторить для каждого свойства, которое должно быть изменено → Apply}* – повторить для создания остальных наборов свойств → [OK] |
| 2. Определение или создание стереотипа компонента | Окно Browser: <Значок диаграммы компонентов> → DbClick → Окно диаграммы: <Требуемый компонент> → RClick → Open Specification → Component Specification → Stereotype = {Новое значение стереотипа Выбранное из списка} → [OK] |
| 3. Создание заголовка и тела компонента | Окно Browser: <Значок диаграммы компонентов> → DbClick → Окно диаграммы: <Требуемый компонент> → RClick → Open Specification → Component Specification → General → Stereotype = {Package Specification – для файла заголовка компонента Package Body – для файла, содержащего тело кода компонента} → [OK] |
| 4. Выбор языка программирования для компонента | {Окно Browser: Окно диаграммы:} <Требуемый компонент> → RClick → Open Specification → Component Specification → General → Language = C++ → [OK] |
| 5. Отнесение класса к компоненту | Окно Browser: → <Значок с компонентами файлов> → DbClick → <Требуемый класс> → |

| | |
|------------|---|
| компоненту | Click_Y → Окно диаграммы: <Требуемый компонент> → Click_Λ |
|------------|---|

Окончание табл. 18.1

| Операция | Рекомендуемая последовательность действий. Примечания |
|--|---|
| 6. Связывание набора свойств с элементом модели | {Окно Browser: Окно диаграммы:} <Элемент модели> → RClick → Open Specification → <Окно спецификации элемента> → C++ → Set = <Требуемый набор свойств> → [OK] |
| 7. Переопределение свойств элемента модели | {Окно Browser: Окно диаграммы:} <Элемент модели> → RClick → Open Specification → <Окно спецификации элемента> → C++ → Set = <Требуемый набор свойств> → 1{Model Properties → <Выбрать свойство и щелкнуть в пределах столбца Value> → {Ввести новое значение свойства Выбрать из списка}}* – повторить для каждого свойства, которое должно быть изменено → [Override] → [OK] |
| 8. Генерация кода | {Окно Browser: Окно диаграммы:} <Выбрать компонент или их группу> → Меню: Tools → C++ → Code Generation |
| 9. Информация о результатах генерации | Окно диалога Code Generation Status |
| 10. Просмотр информации об ошибках и предупреждениях | Меню: View → Log |

Таблица 18.2

Операции к восстановлению модели по коду C++

| Операция | Рекомендуемая последовательность действий. Примечания |
|--|--|
| 1. Создание проекта анализатора C++ | Меню: Tools → C++ → Reverse Engineering → Rational Rose C++ Analyzer → File → New |
| 2. Создание описания проекта анализатора C++ | Rational Rose C++ Analyzer → [Caption] → Caption = <Строка описания проекта> → [OK] |
| 3. Создание списка каталогов проекта анализатора C++ | Rational Rose C++ Analyzer → [Directories] → Project Directory List → 1{Directory Structure = <Выбранный каталог в качестве текущего> → DbClick → {Add Current – для включения текущего каталога в список каталогов Add Subdiry – для включения текущего каталога с подкаталогами первого уровня Add Hierarchy – для включения текущего подкаталога со |

| | |
|--|---|
| | всеми подкаталогами})* – выполнить для каждого каталога, включаемого в список файлов анализа → [OK] |
|--|---|

Окончание табл. 18.2

| Операция | Рекомендуемая последовательность действий. Примечания |
|---|---|
| 4. Создание списка базовых проектов анализатора C++ | Rational Rose C++ Analyzer → [Bases] → Base Projects → 1{Directories = <Выбранный каталог с файлами базовых проектов> → 1{<File Name = <Выбранный файл проекта> → Add}* – повторить для каждого требуемого базового проекта}* – повторить для каждого каталога с файлами требуемых базовых проектов → [OK] |
| 5. Создание списка расширений имен файлов проекта анализатора C++ | Rational Rose C++ Analyzer → [Extensions] → Project File Extensions → 1{New Extensions = <Введенная либо выбранная из списка строка расширения> → [Add]}* – повторить для каждого расширения имени файла → [OK] |
| 6. Создание списка файлов проекта анализатора C++ | Rational Rose C++ Analyzer → [Files] → Project Files → 1{Project Directory list = <Выбранный требуемый каталог> → Files Not in list = <Выбранный файл> → {[Add Selected] [Add All]}* – повторить для каждого каталога из списка каталога проекта → [OK]. Список Files in list можно редактировать в режимах Remove Selected или Remove All, а Project Directory list – в режимах Add Current, Add Subdirs, Add Hierarchy или Remove Dirs |
| 7. Задание категории файла проекта анализатора C++ | Rational Rose C++ Analyzer → Files = <Указанный в списке файл> → Меню: Action → Set Type = {Type 1 Type 2 Type 3} |
| 8. Обработка файла средствами анализатора C++ | Rational Rose C++ Analyzer → Files = <Указанный в списке файл> → Меню: Action → Set Type = {Type 1 Type 2 Type 3} → Rational Rose C++ Analyzer → Files = <Указанный в списке файл> → Меню: Action → {Analyze Code Cycle} |
| 9. Получение информации об ошибках | {Окно протокола Log: Rational Rose C++ Analyzer → Files → <Выбранный файл> → DbClick} |
| 10. Экспорт данных анализатора C++ в модель Rational Rose | Rational Rose C++ Analyzer → Files = <Выбранные файлы для экспорта> → Меню: Action → Export to Rose → Option Set = <Требуемый набор опций экспорта> → [OK] |
| 11. Обновление модели | <Обновляемая модель> → Меню: File → Update → |

| | |
|---------------|---|
| Rational Rose | Update Model → <Нужный каталог> → <Требуемый red-файл> → [OK] |
|---------------|---|

Информация по генерации кода Visual Studio приведена в табл. 18.3, а по обратному восстановлению кода Visual Studio – в табл. 18.4.

Таблица 18.3

Операции к генерации кода Visual Studio

| Операция | Рекомендуемая последовательность действий. Примечания |
|---|--|
| 1. Выбор языка программирования для компонента | {Окно Browser: Диаграмма компонентов:} <Требуемый компонент> → RClick → Open Specification → Component Specification → General → Language → <Выбранный язык {Visual Basic Visual C++}> → [OK] |
| 2. Отнесение класса к компоненту | 1{Окно Browser: Диаграмма компонентов:} <Требуемый компонент> → RClick → Open Specification → Component Specification → Realises → {<Требуемый класс> → RClick → Assign}* – для каждого класса, подлежащего сопоставлению с компонентом → [OK] |
| 3. Активизация приложения Model Assistant Tool для задания свойств кода | {Окно Browser: Окно диаграммы классов:} <Значок класса> → RClick → Model Assistant |
| 4. Активизация приложения Code Update Tool для генерации кода | {Окно Browser: Диаграмма компонентов:} → <Значок компонента> → RClick → Update Code |
| 5. Информация об ошибках | Итоговое (Summary) окно приложения Code Update Tool |

Таблица 18.4

Операции к восстановлению модели по коду Visual Studio

| Операция | Рекомендуемая последовательность действий. Примечания |
|---|--|
| 1. Обновление или создание модели средствами приложения Model Update Tool | Меню: Tools → {Visual Basic Visual C++} → Update Model from Code |
| 2. Информация об ошибках | Итоговое (Summary) окно приложения Code Update Tool |

Генерация исходного кода при разработке ПО «Информационная система обработки полиграфических заказов».

Генерируемый код для классов.

Начнем с анализа программного кода, генерируемого для типичного класса. В процессе генерации класс объектной модели становится классом C++. Для каждого класса генерируется программная конструкция примерно следующего вида:

```
Class TheClass
{
public:
TheClass();
TheClass();
};
```

Однако в программном коде генерируется еще и большой объем дополнительной информации. Рассмотрим завершенные файлы заголовка и реализации. В генерируемом коде отражаются все атрибуты, операции и отношения класса. К основным элементам, генерируемым для каждого класса, относятся:

- имя класса;
- видимость класса;
- конструктор для класса (необязателен);
- деструктор для класса (необязателен);
- операции Get() и Set() для каждого атрибута (необязательны);
- текстовое описание (документация) класса;
- атрибуты;
- операции;
- отношения.

Для каждого класса модели генерируются два файла C++: файл заголовка и файл реализации, причем оба они именуются в соответствии с именем класса. Например, для класса Employee генерируются файлы Employee.h и Employee.cpp.

При генерации программного кода Rational Rose использует пакетную структуру, организованную в компонентном представлении модели, для создания соответствующих каталогов. Каталог создается для каждого пакета модели. Внутри каждого каталога, создаваемого в Rational Rose, будут находиться файлы .cpp и .h для классов данного пакета. Если компоненты и пакеты не были созданы в компонентном представлении, Rational Rose применяет для организации структуры каталогов пакетную структуру логического представления.

Значительный объем информации модели Rational Rose используется в процессе генерации напрямую. Атрибуты, операции,

связи и имена всех классов оказывают прямое воздействие на генерируемый программный код. Другие поля модели, например текстовое описание класса, не влияют непосредственно на программу. Значения этих полей в генерируемом программном коде становятся комментариями.

В табл. 18.5 представлены поля, доступные в окне Class Specification, и отмечены те из них, которые напрямую воздействуют на генерируемый программный код.

Таблица 18.5

Поля, доступные в окне Class Specification

| Поле | Воздействие на программный код |
|------------------|---|
| Name | Имя в модели становится именем класса |
| Type | Напрямую влияет на тип создаваемого класса |
| Stereotype | Не действует |
| Export Control | Непосредственно влияет на область видимости класса |
| Documentation | Комментарий |
| Cardinality | Не действуют |
| Space | |
| Persistence | |
| Concurrency | |
| Abstract | Создает абстрактный класс |
| Formal Arguments | Формальные аргументы включаются в программный код для параметризованного класса |
| Operations | Генерируются в программном коде |
| Attributes | |
| Relationships | |

Посмотрим, какой программный код генерируется для следующего класса:

| ORDER |
|---|
| - ORDERNumber:int |
| - DepartureDate:date |
| - DepartureCity:string |
| - ArrivalCity:string |
| + AddPassenger(PassengerID:int):boolean |

```
+ RemovePassenger(PassengerID:int) boolean
+ CancelORDER():int
```

Генерируемый файл заголовка.

Далее приведен фрагмент программы, который является файлом заголовка, генерируемым для данного класса.

```
#ifndef ORDER_H_INCLUDED_C6AD4E5A
#define
ORDER_H_INCLUDED_C6AD4E5A
///ModelId=39528510031C///Documentation
/// This class holds information about airline ORDERS.
class ORDER
{
public:
///ModelId=3952853300EC
boolean AddPassenger(int PassengerID);
///ModelId=3952853B013D
boolean RemovePassenger(int PassengerID);
///ModelId=3952854302B1
int CancelORDER();
private:
///ModelId=395285160108
int ORDERNumber;
///ModelId=395285190365
date DepartureDate;
///ModelId=3952851E0037 string DepartureCity;
///ModelId=3952852B009A string ArrivalCity;
};
#endif /* ORDER_H_INCLUDED_C6AD4E5A */
```

Начнем с аннотации, добавленной в исходный код средой Rational Rose. Включены идентификаторы модели, поэтому можно модифицировать и регенерировать код (при повторном проектировании) без перезаписи изменений. Идентификаторы модели необязательны. Их включение или выключение выполняется параметром Never Generate Model IDs (Никогда не генерировать идентификаторы модели) на вкладке Style (Стиль) окна спецификации компонента ANSI C++.

Сгенерированный файл будет содержать заголовочную информацию для атрибутов и операций класса, а также заголовки для

конструктора, деструктора, конструктора копирования и других стандартных методов. Для включения этих заголовков открывают окно ANSI C++ Class Customization (Настройка класса ANSI C++) и указывают генерируемый(е) метод(ы), либо используют помощник Model Assistant для Visual C++ (рис. 18.1, 18.2).

Из рисунков видно, что Rational Rose включает для каждой операции видимость, параметры, типы данных параметров и возвращаемый тип данных. Хотя среда Rational Rose не может сгенерировать сам код операции, формируется «скелетный» каркас для дальнейшего программирования. Можно сгенерировать и код операции, если добавить код по умолчанию в свойство Initial Code Body определенной операции.

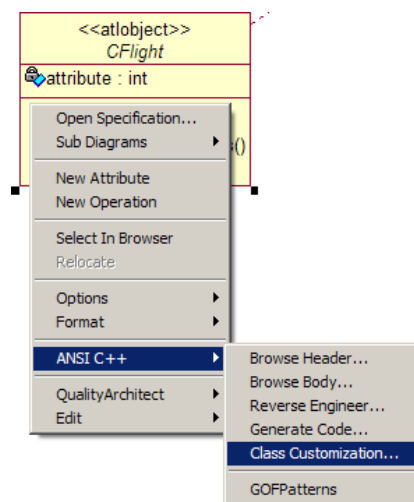


Рис. 18.1. Окно ANSI C++ Class Customization

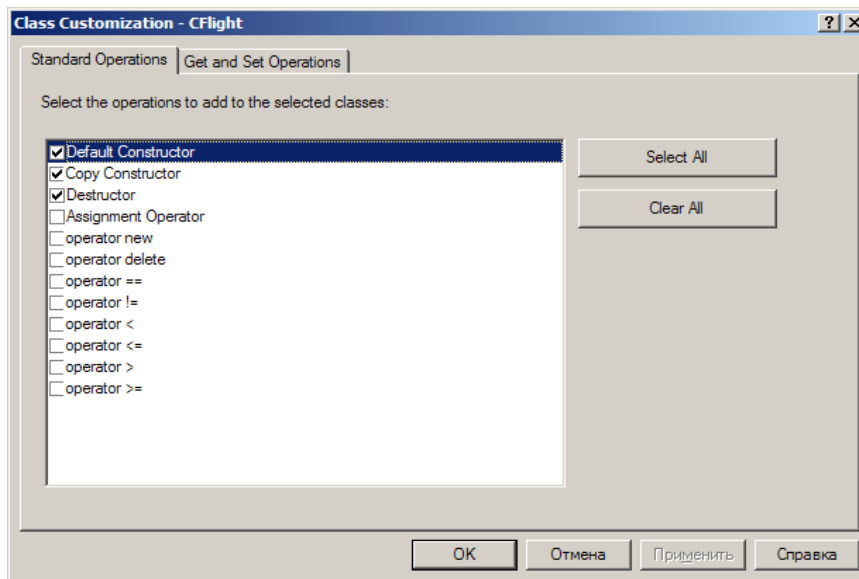


Рис. 18.2. Помощник Model Assistant для Visual C++

Генерируемый файл реализации.

Средой Rational Rose также генерируется файл реализации с расширением по умолчанию .cpp. Ниже показан сгенерированный файл реализации для рассмотренного выше заголовочного файла.

```
#include "c:/ORDER.h"
///

```

Среда Rational Rose добавила в сгенерированный код операции видимость, параметры, типы данных параметров и возвращаемый тип данных. Если указать режим генерации конструктора, деструктора или иных стандартных методов (щелкают правой кнопкой на классе и выбирают ANSI C++ Code Customization), то соответствующие фрагменты будут добавлены в код. Для Visual C++ можно использовать

помощника Model Assistant, чтобы указать режимы генерации отдельных методов.

Генерируемый код для атрибутов.

Кроме самих классов, среда Rational Rose генерирует атрибуты для классов. Для каждого атрибута добавляется:

- видимость;
- тип данных;
- значение по умолчанию;
- операция Get (необязательна);
- операция Set (необязательна).

Для данного атрибута мы получим в Rational Rose примерно такой код:

```
Class TheClass
{
public:
int PublicAttribute;
int GetPublicAttribute();
int GetProtectedAttribute();
int GetPrivateAttribute();
void Set_PublicAttribute(int value);
void Set_ProtectedAttribute(int value);
void Set_PrivateAttribute(int value);
protected:
int ProtectedAttribute;
private:
int PrivateAttribute;
};
```

Для дальнейшего совершенствования кода можно включить комментарии и идентификаторы Rational Rose, которые попадут в полные файлы заголовка и реализации. Рассмотрим код, генерируемый для следующего класса:

| ORDER |
|--|
| – ORDERNumber::int |
| – DepartureDate:date |
| – DepartureCity:string |
| – ArrivalCity:string |
| + AddPassenger(PassengerID:int):boolean |
| + RemovePassenger(PassengerID:int):boolean |
| + CancelORDERQ:int |

```
«const» + Get_ORDERNumber():int
+ Set_ORDERNumber(left:int):void
```

В этом примере было открыто окно ANSI C++ Class Customization и указана генерация методов Get и Set для атрибута ORDERNumber. Для Visual C++ это можно сделать в помощнике Model Assistant. Заголовочный файл станет таким:

```
#ifndef ORDER_H__INCLUDED_C6ABFE6A
#define ORDER_H__INCLUDED_C6ABFE6A
///ModelId=39541274036B
class ORDER
{
public:
///ModelId=3954129803B3
boolean AddPassenger(int PassengerID);
///ModelId=395412A00093
boolean RemovePassenger(int PassengerID);
///ModelId=395412A80121
int CancelORDER();
///ModelId=395413D80055
int Get_ORDERNumber() const;
///ModelId=395413D800BA
void Set_ORDERNumbe (int left);
private:
///ModelId=3954127801F4
int ORDERNumber:
///ModelId=3954128202DF
date DepartureDate;
///ModelId=395412860122
string DepartureCity;
///ModelId=3954128E0097
string ArrivalCity;
};
```

В файл реализации тоже попадут методы Get и Set, причем Rational Rose включит не только сигнатуру метода, но и код самого метода. Файл реализации для класса ORDER будет таким:

```
#include "c:/ORDER.h"
///ModelId=3954129803B3
boolean ORDER::AddPassenger(int PassengerID)
```

```

{
}
///ModelId=395412A00093
boolean ORDER::RemovePassenger(int PassengerID)
{
}
///ModelId=395412A80121
int ORDER::CancelORDER()
{
}
///ModelId=395413D80055
int ORDER::Get_ORDERNumber() const
{
return ORDERNumber;
}
///ModelId=395413D800BA
void ORDER::Set_ORDERNumber(int left)
{
ORDERNumber=left;
}

```

Генерируемый код для операций.

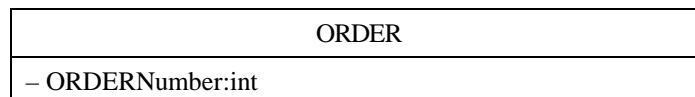
Rational Rose генерирует код для всех операций класса, причем добавляет имя операции, параметры, типы данных параметров и возвращаемый тип данных. Для каждой операции генерируется примерно такой код:

```

Class TheClass
{
public:
void PublicOperation();
protected:
void ProtectedOporation();
private:
void PrivateOperation();
};

```

Рассмотрим код, сгенерированный для класса:



| |
|--|
| - DepartureData:date - DepartureCity:string - ArrivalCity:string |
| + AddPassenger(PassengerID:int):boolean + RemovePassenger(PassengerID:int):boolean + CancelORDER():int |

В заголовочный файл среда Rational Rose добавит сигнатуру операций:

```

#ifndef ORDER_H_INCLUDED_C6ABEA96
#define ORDER_H_INCLUDED_C6ABEA96
///ModelId=39641274036B
class ORDER
{
public:
///ModelId=3954129803B3
boolean AddPassenger(int PassengerID);
///ModelId=396412A00093
boolean RemovePassenger(int PassengerID);
///ModelId=396412A80121
///Documentation
/// The CancelORDER operation will cancel all reservations
/// for the ORDER, notify all passengers with reservations
/// end dissable future reservations for the ORDER
int CancelORDER();
private:
///ModelId=3954127801F4
int ORDERNumber;
///ModelId=39541282020F
date DepartureDate;
///ModelId=395412860122
string DepartureCity;
///ModelId=3954128E0097
string ArrivalCity;
};
#endif /* ORDER_H_INCLUDED,,C6ABEA96 */

```

В коде сгенерирована полная сигнатура (подпись) операции. Добавляется также вся документация, введенная в классе (в виде

комментариев). Если включена информация о протоколе операции, квалификаторах, исключениях, времени, пространстве, предусловиях, семантике или постусловиях, то все это также попадает в сгенерированный код.

Кроме того, Rational Rose генерирует код операций в файле реализации. Ниже приводится файл реализации для класса ORDER.

```
#include "c:/ORDER.h"
///
```

Rational Rose включает все операции в файл реализации. Разработчикам следует отредактировать этот файл и ввести код операции между открывающей и закрывающей фигурными скобками.

18.3. Редактор VISIO и кодогенерация

VISIO как инструмент генерации исходного кода по диаграмме классов уступает Rational Rose только в смысле предоставляемого сервиса, но тем не менее создание каркаса программируемой системы поддерживает. Выбрав в VISIO Меню: File → New → Software → UML Model Diagram, можно воспользоваться, как уже указывалось выше, появившимися трафаретами для построения модели программируемой системы. Затем, выделив на диаграмме классов необходимые элементы для генерации исходного кода, следует выбрать в появившейся секции UML опцию Code → Generate, как это показано на рис. 18.3.

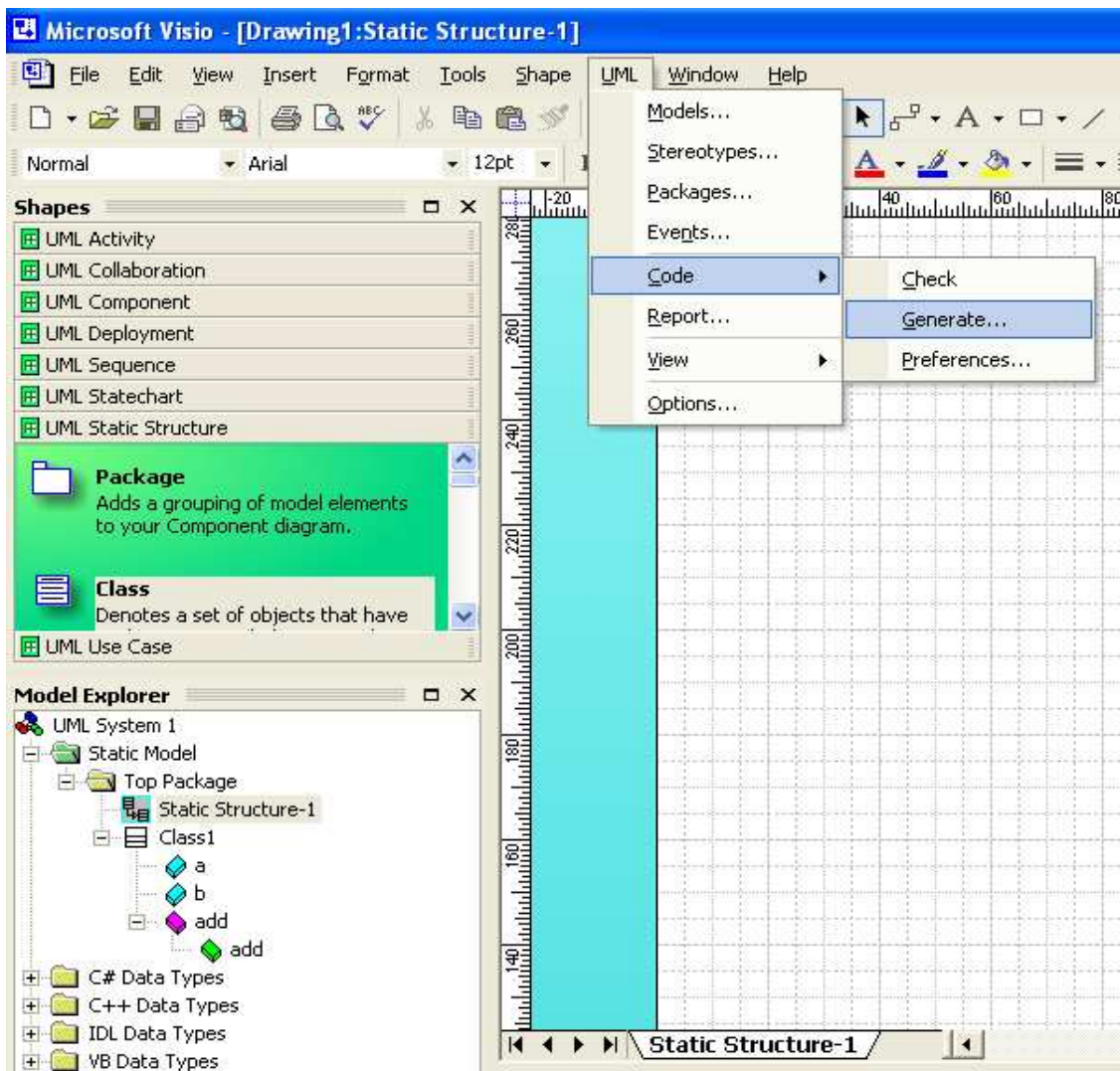


Рис. 18.3. Окно программы в момент выбора Меню: UML → Code → Generate

Вопросы для самоконтроля

1. Как создать исходный код VC по диаграмме классов в Rational Rose?
2. Какова структура создаваемого кода?
3. Что такое Model Assistant?
4. Какова структура MFC-приложения?
5. Какие действия следует предпринять для обновления модели по исходному коду?
6. Можно ли создать исходный код VC в VISIO?
7. Какая диаграммная техника является основой кодогенерации?

ЗАКЛЮЧЕНИЕ

В учебном пособии рассматривается современный подход к разработке программного обеспечения (ПО) для случая, когда созданию исходного кода программы предшествуют определение ее места в контексте AS – IS и TO – BE, построение логической и физической моделей предмета разработки при использовании современных средств САПР ПО. В первом разделе пособия перечислены основные этапы жизненного цикла (ЖЦ) ПО с акцентом на критические этапы и успешное их выполнение с применением CASE-пакетов AllFusion Modeling Suite, Rational Rose и графического редактора VISIO. В последующих разделах излагается последовательность моделирования на примере разработки ПО в области полиграфии, с построением моделей AS – IS и TO – BE, логической и физической моделей данных, логической и физической моделей ПО с последующей кодогенерацией. При этом подход к раскрытию содержания разделов тесно связан с их названием. Например, разделы, посвященные той либо иной диаграммной технике, содержат такие сведения, как:

- назначение и основные элементы диаграммы;
- особенности ее построения с использованием конкретного CASE-средства и руководство пользователю по выполнению операций, связанных с построением диаграммы;
- пример применения диаграммной техники при работе над моделью предмета разработки – ПО, связанного с издательско-полиграфическим комплексом;
- указания на возможность использования в качестве альтернативного инструмента графического редактора VISIO.

Несмотря на то, что материал учебного пособия адресован разработчикам ПО издательско-полиграфического комплекса, оно может, по мнению авторов, найти более широкое применение, а именно, в разработке ПО иного назначения.

Кроме того, разделы, посвященные IDEF-техникам и инструментарию их поддержки, могут быть использованы экономистами, а также представителями других специальностей, прибегающими к упомянутым диаграммным техникам.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Калянов, Г. Н. CASE: структурный системный анализ (автоматизация и применение) / Г. Н. Калянов. – М.: ЛОПН, 1996. – 242 с.
2. Калянов, Г. Н. CASE-технологии. Консалтинг в автоматизации бизнес-процессов / Г. Н. Калянов. – 3-е изд. – М.: Горячая линия – Телеком, 2002. – 320 с.
3. Маклаков, С. В. ВРwin и ERwin. CASE-средства разработки информационных систем / С. В. Маклаков. – 2-е изд., испр. и доп. – М.: ДИАЛОГ-МИФИ, 2001. – 304 с.
4. Маклаков, С. В. Создание информационных систем с AllFusion Modeling Suite / С. В. Маклаков. – М.: ДИАЛОГ-МИФИ, 2005. – 288 с.
5. Черемных, С. В. Моделирование и анализ систем. IDEF-технологии: практикум / С. В. Черемных, И. О. Семенов, В. С. Ручкин. – М.: Финансы и статистика, 2002. – 192 с.
6. Черемных, О. С. Стратегический корпоративный реинжиниринг: процессно-стоимостной подход к управлению бизнесом / О. С. Черемных, С. В. Черемных. – М.: Финансы и статистика, 2005. – 736 с.
7. Харрингтон, Д. Проектирование реляционных баз данных. Просто и доступно / Д. Харрингтон; пер. с англ. – М.: Лори, 2002. – 230 с.
8. Харрингтон, Д. Проектирование объектно-ориентированных баз данных / Д. Харрингтон; пер. с англ. – М.: ДМК Пресс, 2001. – 272 с.
9. Буч, Г. Объективно-ориентированный анализ и проектирование с примерами приложений на С++ / Г. Буч; пер. с англ. – 2-е изд. – М.: Бином, СПб.: Невский диалект, 1999. – 560 с.
10. Буч, Г. Язык UML: Руководство пользователя / Г. Буч, Д. Рамбо, А. Джекобсон; пер. с англ. – М.: ДМК, 2000. – 432 с.
11. Бадд, Т. Объектно-ориентированное программирование в действии / Т. Бадд; пер. с англ. – СПб.: Питер, 1997. – 464 с.
12. Леоненков, А. В. Самоучитель UML / А. В. Леоненков. – СПб.: БХВ-Петербург, 2001. – 304 с.
13. Шмуллер, Дж. Освой самостоятельно UML за 24 часа / Дж. Шмуллер; пер. с англ. – 2-е изд. – М.: Вильямс, 2002. – 352 с.
14. Фаулер, М. UML. Основы / М. Фаулер, К. Скотт; пер. с англ. – СПб.: Символ-Плюс, 2002. – 192 с.
15. Кватрани, Т. Визуальное моделирование с помощью Rational Rose 2002 и UML / Т. Кватрани; пер. с англ. – М.: Вильямс, 2003. – 192 с.

16. Трофимов, С. А. CASE-технологии: практическая работа в Rational Rose / С. А. Трофимов. – 2-е изд. – М.: Бином-Пресс, 2002. – 288 с.
17. Трофимов, С. А. Rational XDE для Visual Studio. NET / С. А. Трофимов. – М.: Бином-Пресс, 2004. – 304 с.
18. Технология программирования: Моделирование программных систем: метод. указания и задания к лабораторным работам / сост. Л. Ф. Дробушевич. – Минск: БГУ, 2003. – 66 с.
19. Федотова, Д. Э. CASE-технологии: практикум / Д. Э. Федотова, Ю. Д. Семенов, К. Н. Чижик. – М.: Горячая линия – Телеком, 2003. – 160 с.
20. Карпов, Б. Microsoft VISIO 2000: Краткий курс / Б. Карпов, Н. Мирошниченко. – СПб.: Питер, 2001. – 256 с.

СОДЕРЖАНИЕ

| | |
|---|----|
| ПРЕДИСЛОВИЕ | 3 |
| УСЛОВНЫЕ ОБОЗНАЧЕНИЯ..... | 5 |
| 1. САПР В КОНТЕКСТЕ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ | 6 |
| 1.1. Жизненный цикл программного обеспечения и его критические этапы..... | 6 |
| 1.2. САПР программного обеспечения в жизненном цикле про- граммного обеспечения | 8 |
| 1.3. Инструментальная поддержка автоматизации разработки программного обеспечения | 9 |
| 1.4. Концептуальные основы CASE-технологий..... | 12 |
| 1.5. Состав, структура и функциональные особенности CASE- средств | 13 |
| 1.6. Классификация CASE-средств | 14 |
| 2. МОДЕЛИРОВАНИЕ БИЗНЕС-ПРОЦЕССОВ | 19 |
| 2.1. Нотация IDEF0 и моделирование процессов | 19 |
| 2.2. Инструментальная среда AllFusion Process Modeler | 21 |
| 2.3. Разработка IDEF0-моделей процессов в AllFusion Process Modeler | 22 |
| 2.4. Редактор VISIO и нотация IDEF0 | 29 |
| 3. НОТАЦИЯ DFD И МОДЕЛИРОВАНИЕ ПРОЦЕССОВ | 32 |
| 3.1. Элементы нотации DFD | 32 |
| 3.2. Разработка DFD-моделей процессов в AllFusion Process Modeler..... | 33 |
| 3.3. Редактор VISIO и нотация DFD | 36 |
| 4. НОТАЦИЯ IDEF3 И МОДЕЛИРОВАНИЕ ПРОЦЕССОВ | 39 |
| 4.1. Элементы нотации IDEF3 | 39 |
| 4.2. Разработка IDEF3-моделей процессов в среде AllFusion Process Modeler | 41 |
| 4.3. Редактор VISIO и нотация IDEF3 | 45 |
| 5. ОЦЕНИВАНИЕ ПРОЦЕССОВ МОДЕЛИ AS – IS В ALLFUSION PROCESS MODELER И ПОСТРОЕНИЕ МОДЕЛИ TO – BE | 46 |
| 5.1. Оценивание бизнес-процессов в AllFusion Process Modeler ... | 46 |

| | |
|---|----|
| 5.2. Построение модели ТО – ВЕ при разработке программного обеспечения «Информационная система обработки полиграфических заказов» | 52 |
| 6. МОДЕЛИРОВАНИЕ ДАННЫХ | 61 |
| 6.1. Построение ER-модели данных..... | 61 |
| 6.2. Этапы разработки диаграммы «сущность – связь» | 62 |
| 7. ЛОГИЧЕСКОЕ МОДЕЛИРОВАНИЕ ДАННЫХ..... | 68 |
| 7.1. Инструментальная среда AllFusion ERwin Data Modeler | 68 |
| 7.2. Разработка логической модели данных в AllFusion ERwin Data Modeler | 69 |
| 7.3. Построение логической модели данных при разработке программного обеспечения «Информационная система обработки полиграфических заказов» | 71 |
| 7.4. Редактор VISIO и логическое моделирование данных..... | 74 |
| 8. ФИЗИЧЕСКОЕ МОДЕЛИРОВАНИЕ ДАННЫХ..... | 77 |
| 8.1. Разработка физической модели данных в AllFusion ERwin Data Modeler | 77 |
| 8.2. Построение физической модели данных при разработке программного обеспечения «Информационная система обработки полиграфических заказов» | 79 |
| 8.3. Редактор VISIO и физическое моделирование данных | 82 |
| 9. ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ ПОДХОД К РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ..... | 84 |
| 9.1. Понятия ООА, ООД и ООР | 84 |
| 9.2. Объектная модель | 84 |
| 9.3. Объектно-ориентированный процесс разработки программного обеспечения..... | 86 |
| 9.4. Введение в UML и инструментарий его поддержки..... | 87 |
| 9.5. Инструментальная среда Rational Rose | 88 |
| 10. МОДЕЛЬ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ И ЕЕ РАЗРАБОТКА..... | 91 |
| 10.1. Элементы нотации диаграммы прецедентов и особенности ее построения..... | 91 |

| | | |
|-------|---|-----|
| 10.2. | Разработка диаграммы прецедентов в Rational Rose..... | 94 |
| 10.3. | Создание модели прецедентов при разработке программного обеспечения «Информационная система обработки полиграфических заказов»..... | 96 |
| 10.4. | Редактор VISIO и диаграмма вариантов использования..... | 97 |
| 11. | ДАЛЬНЕЙШЕЕ РАЗВИТИЕ МОДЕЛИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ..... | 100 |
| 11.1. | Диаграмма состояний и ее разработка..... | 100 |
| 11.2. | Разработка диаграммы состояний в Rational Rose..... | 102 |
| 11.3. | Редактор VISIO и диаграмма состояний..... | 104 |
| 12. | ДИАГРАММА ДЕЯТЕЛЬНОСТИ И ЕЕ РАЗРАБОТКА..... | 106 |
| 12.1. | Элементы нотации диаграммы деятельности и особенности ее построения..... | 106 |
| 12.2. | Разработка диаграммы деятельности в Rational Rose | 108 |
| 12.3. | Редактор VISIO и диаграмма деятельности | 110 |
| 13. | ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ И ЕЕ РАЗРАБОТКА.. | 112 |
| 13.1. | Элементы нотации диаграммы последовательности и особенности ее построения..... | 112 |
| 13.2. | Разработка диаграммы последовательности в Rational Rose..... | 114 |
| 13.3. | Редактор VISIO и диаграмма последовательности | 116 |
| 14. | ДИАГРАММА КООПЕРАЦИЙ И ЕЕ РАЗРАБОТКА..... | 118 |
| 14.1. | Элементы нотации диаграммы коопераций и особенности ее построения..... | 118 |
| 14.2. | Разработка диаграммы коопераций в Rational Rose | 122 |
| 14.3. | Редактор VISIO и диаграмма коопераций | 123 |
| 15. | ДИАГРАММА КЛАССОВ И ЕЕ РАЗРАБОТКА | 126 |
| 15.1. | Элементы нотации диаграммы классов и особенности ее построения..... | 126 |
| 15.2. | Разработка диаграммы классов в Rational Rose..... | 129 |
| 15.3. | Редактор VISIO и диаграмма классов..... | 137 |
| 16. | ДИАГРАММА КОМПОНЕНТОВ И ЕЕ РАЗРАБОТКА | 138 |
| 16.1. | Элементы нотации диаграммы компонентов и особенности ее построения..... | 138 |

| | |
|---|-----|
| 16.2. Разработка диаграммы компонентов в Rational Rose | 140 |
| 16.3. Редактор VISIO и диаграмма компонентов | 143 |
| 17. ДИАГРАММА РАЗВЕРТЫВАНИЯ И ЕЕ РАЗРАБОТКА | 145 |
| 17.1. Элементы нотации диаграммы развертывания и особен- ности ее построения | 145 |
| 17.2. Разработка диаграммы развертывания в Rational Rose | 147 |
| 17.3. Редактор VISIO и диаграмма развертывания | 148 |
| 18. КОДОГЕНЕРАЦИЯ | 150 |
| 18.1. Понятие генерации исходных текстов программ | 150 |
| 18.2. Генерация исходного кода в Rational Rose..... | 150 |
| 18.3. Редактор VISIO и кодогенерация | 164 |
| ЗАКЛЮЧЕНИЕ | 166 |
| СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ..... | 167 |

Учебное издание

Бугай Осип Викентьевич
Юденков Виктор Степанович

**САПР ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ
ИЗДАТЕЛЬСКО-ПОЛИГРАФИЧЕСКОГО КОМПЛЕКСА**

Учебное пособие

Редактор *Е. С. Ватеичкина*
Верстка *И. А. Ткаченко*

Подписано в печать 04.06.2008. Формат 60×84¹/₁₆.
Бумага офсетная. Гарнитура Таймс. Печать офсетная.
Усл. печ. л. 10,1. Уч.-изд. л. 9,0.
Тираж 150 экз. Заказ .

Учреждение образования
«Белорусский государственный технологический университет».
220006. Минск, Свердлова, 13а.
ЛИ № 02330/0133255 от 30.04.2004.

Отпечатано в лаборатории полиграфии учреждения образования
«Белорусский государственный технологический университет».
220006. Минск, Свердлова, 13.

ЛП № 02330/0056739 от 22.01.2004.