администратор информационной системы и члены команды, разрабатывающей приложение, также являются заинтересованными лицами, так как они будут отвечать за разработку и сопровождение системы. Они также, как и пользователи, будут зависеть от поведения системы. Результаты выявления пользователей и заинтересованных лиц новой системы ввода заказов на покупку представлены в таблица 2.3.

Таблица 2.3 - Пользователи и лица, заинтересованные в новой системе

Пользователи	Другие заинтересованные лица
Служащие, занимающиеся вводом заказов Руководитель отдела приема заказов Контроль производства Служащий, выписывающий счета	Администратор информационной системы и команда разработчиков Главный финансист Управляющий производством

2.2.5 Этап 4. Определение границ системы-решения

После того как согласована постановка проблемы и выявлены пользователи и заинтересованные лица, можно перейти к *определению системы,* разрабатываемой для решения данной проблемы. Это важный момент, когда необходимо постоянно помнить как о понимании проблемы, так и о свойствах потенциального решения.

Граница системы описывает оболочку, в которой заключена система (рисунок 2.2). Информация в виде ввода и вывода передается от находящихся вне системы пользователей системе и обратно. Все взаимодействия с системой осуществляются посредством интерфейсов между системой и внешним миром.

Рисунок 2.2 - Отношение ввод/система/вывод

Другими словами, если мы собираемся нечто создать или модифицировать – это часть нашего решения, которая находится внутри границы; если нет – это нечто внешнее по отношению к системе. Таким образом, *мы делим мир на два интересующих нас класса*.

1. Наша система.

2. То, что взаимодействует с нашей системой.

Определим то, что взаимодействует с нашей системой, общим понятием «актанты» (actors). Они выполняют некоторые действия, заставляя систему делать ее работу. Актант изображается простой пиктограммой в виде человечка. Его определение выглядит следующим образом.

Актант – это находящееся вне системы нечто (или некто), взаимодействующее с системой.

С помощью данного понятия мы можем проиллюстрировать границы системы.

Во многих случаях границы системы очевидны. Например, однопользовательский персональный планировщик контактов, работающий на автономной платформе Windows 2000, имеет достаточно хорошо определенные границы. Имеется всего один пользователь и одна платформа. Интерфейсы между пользователем и приложением состоят из диалогов, посредством которых пользователь получает доступ к информации системы, и неких выходных сообщений и коммуникационных путей, которые система использует для документирования или передачи этой информации.

Для системы ввода заказов из нашего примера, которая должна быть объединена с уже существующей информационной системой компании, границы не столь очевидны. Аналитик должен определить, будут ли данные использоваться совместно с другими приложениями, должно ли новое приложение распределяться по разным хостам и клиентам, а также кто будет пользователем. Например, должен ли персонал, занятый в производстве, иметь интерактивный доступ к заказам на покупку? Обеспечивается ли контроль качества или функции аудита? Будет ли система выполняться на компьютере-мэйнфрейме или на новом компьютере-клиенте? Должны ли предоставляться специальные отчеты?

Выявление актантов является ключевым аналитическим этапом в анализе проблемы.

Ответы на следующие вопросы помогут их обнаружить.

Кто будет поставлять, использовать или удалять информацию из системы?

Кто будет управлять системой?

Кто будет осуществлять сопровождение системы?

Где будет использоваться система?

Откуда система получает информацию?

Какие внешние системы будут взаимодействовать с системой?

Имея ответы на эти вопросы, аналитик может создать блок-схему, описывающую границы системы, пользователей и другие интерфейсы.

2.2.6. Этап 5. Выявление ограничений, налагаемых на решение

Ограничение уменьшает степень свободы, которой мы располагаем при предложении решения.

Каждое ограничение может значительно сузить нашу возможность создать предполагаемое решение. Следовательно, в процессе планирования необходимо тщательно изучить все ограничения. Многие из них могут даже заставить нас пересмотреть изначально предполагавшийся технологический подход.

Необходимо учитывать, что существуют различные источники ограничений (экономические, технические, политические и т. д.). Ограничения могут быть заданы еще до начала работы («Никакой новой аппаратуры!»), но может получиться, что нам действительно придется их выявлять.

Чтобы их выявить, полезно знать, на что следует обратить внимание.

В таблице 2.4 указаны возможные источники системных ограничений. Приведены в таблице вопросы помогут выявить большую часть ограничений. Часто полезно получить объяснение ограничения, как для того, чтобы убедиться, что вы поняли его назначение, так и для того, чтобы можно было обнаружить (если такое произойдет), что данное ограничение больше не применимо к вашему решению.

После того как ограничения выявлены, некоторые из них станут требованиями к новой системе. Другие ограничения будут оказывать влияние на ресурсы и планы реализации. Именно при анализе проблемы необходимо выявить потенциальные источники ограничений и понять, какое влияние каждое ограничение окажет на область возможных решений.

Возвратимся к нашему примеру. Ограничения, которые могут налагаться на новую систему ввода заказов на покупку, представлены в таблице 2.5.

Таблица 2.4 - Возможные источники ограничений системы

Источник	Образцы вопросов
Экономический	Какие финансовые или бюджетные ограничения следует учесть? Существуют ли соображения, касающиеся себестоимости и ценообразования? Существуют ли вопросы лицензирования?
Политический	Существуют ли внешние или внутренние политические вопросы, влияющие на потенциальное решение? Существуют ли проблемы в отношениях между подразделениями?
Технический	Существуют ли ограничения в выборе технологий? Должны ли мы работать в рамках существующих платформ или технологий? Запрещено ли использование любых новых технологий? Должны ли мы использовать какие-либо закупаемые пакеты программного обеспечения?
Системный	Будет ли решение создаваться для наших существующих систем? Должны ли мы обеспечивать совместимость с существующими решениями? Какие операционные системы и среды должны поддерживаться?
Эксплуатационный	Существуют ли ограничения информационной среды или правовые ограничения? Юридические ограничения? Требования безопасности? Какими другими стандартами мы ограничены?
График и ресурсы	Определен ли график? Ограничены ли мы существующими ресурсами? Можем ли мы привлекать работников со стороны? Можем ли мы увеличить штат? Временно? Постоянно?

Таблица 2.5 - Ограничения, налагаемые на систему ввода заказов на покупку

Источник	Ограничение	Объяснение
Эксплуатационный	Копия данных заказа на покупку должна оставаться в унаследованной базе данных в течение одного года	Риск потери данных слишком велик
Системы и операционные системы	Приложение должно занимать на сервере не более 20 мегабайт	Количество доступной памяти сервера ограничено
Средства, выделенные на оборудование	Система должна быть разработана на существующем сервере	Сокращение издержек и поддержка существующих систем
Средства, выделенные на оплату труда персонала	Фиксированный штат; не привлекать работников со стороны	Фиксированные расходы на зарплату по отношению к текущему бюджету
Технические требования	Должна использоваться новая объектно-ориентирован-ная технология	Мы надеемся, что эта технология повысит производительность и надежность ПО

2.3 Методология ARIS

В теории систем различают структуру системы и ее поведение. Структура описывает статику системы, а поведение описывает динамику. Четыре первых представления в ARIS (организационное, функциональное, представление выходов и представление данных) описывают структуру системы. В процессном представлении устанавливаются связи перечисленных представлений, и описывается динамическое поведение системы. На рисунке 2.3 перечисленные представления сгруппированы в пять частей, образующих здание АРИС.

Функциональное представление содержит описание целей, выполняемых функций, перечень отдельных подфункций, а также общие взаимосвязи и связи подчиненности, которые существуют между функциями, целями и факторами успеха.

Организационное представление описывает взаимодействие пользователей и организационных единиц (ОЕ), а также их связи и имеющие к ним отношение структуры. Организационные модели служат для описания иерархической структуры организации, где группируются субъекты ответственности, ОЕ, должностей и средств, выполняющих работу над объектами, а также для многого другого.

В представлении данных описывают информационную среду предприятия (среду обработки данных), события, сообщения и т. д., где неявно фиксируются также объекты в виде информационных услуг.

Представление процессов (управления) используется для описания связей между тремя предшествующими представлениями. Интеграция этих связей в пределах отдельного представления делает возможным учесть все связи без избыточности. Модели представления процессов описывают отношения между отдельными моделями в рамках всего бизнес-процесса. Это позволяет отслеживать все двусторонние и многосторонние отношения между моделями различных видов, а также полностью описать бизнес-процесс.

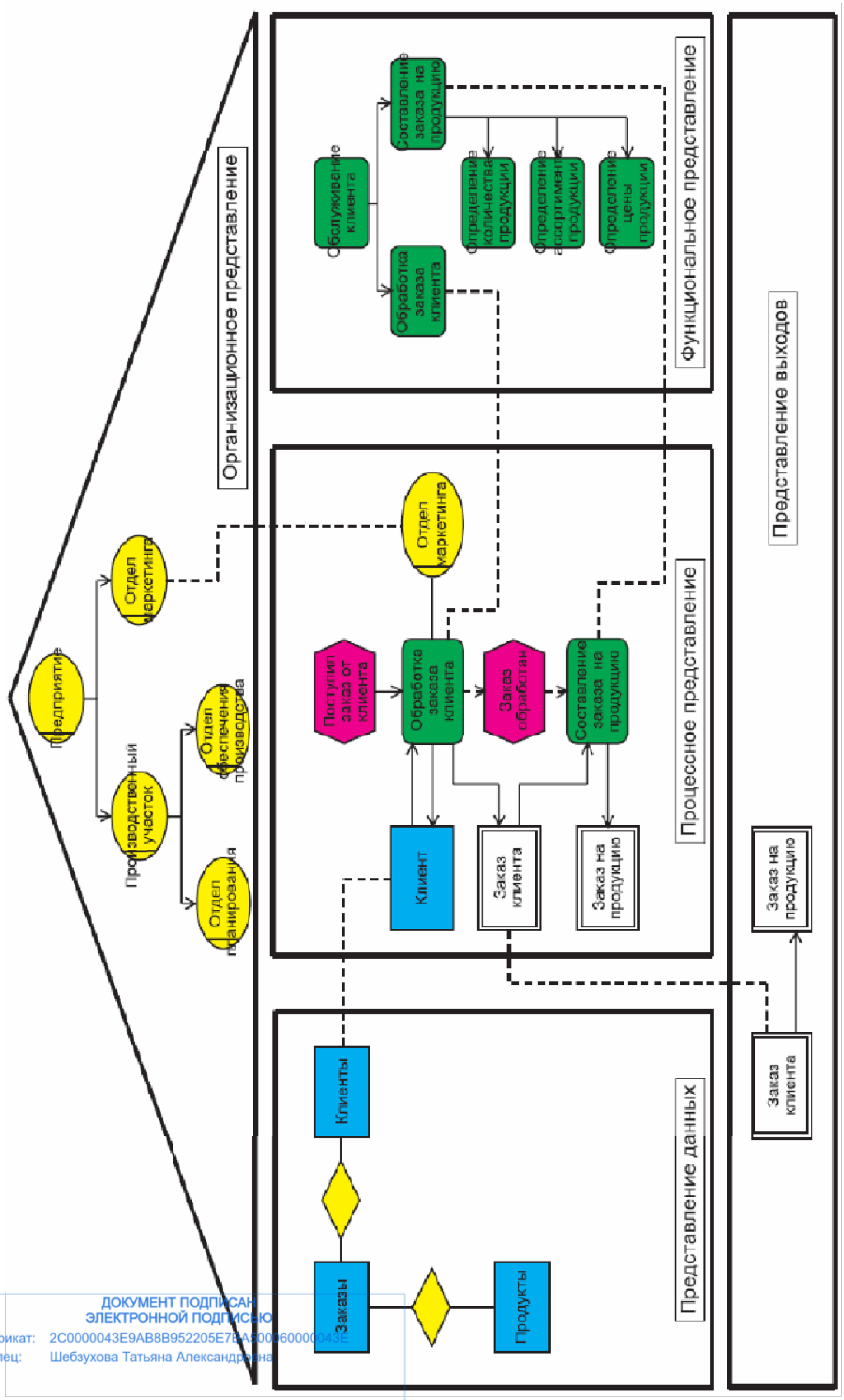


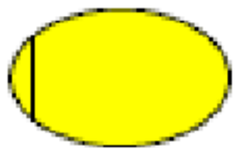
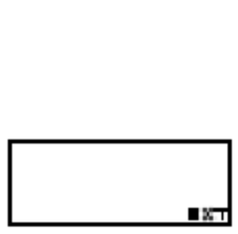
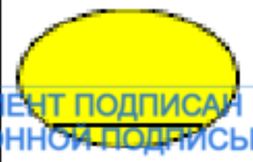
Рисунок 2.3. «Здание» ARIS. Представления

2.3.1 Организационная модель

Организационная диаграмма позволяет всесторонне описать организационно-штатную структуру предприятия. Организационные диаграммы показывают имеющиеся организационные подразделения (как исполнители функций) и их взаимозависимости в соответствии с выбранными критериями структурирования. Отдельные организационные единицы соединяются связями для указания иерархии.

Модель организационной структуры предприятия может содержать следующие структурные элементы: организационная единица, должность или позиция, личность, группа и др. (таблица 2.6).

Таблица 2.6 - Элементы организационной диаграммы

№ п/п	Графическая нотация	Наименование	Описание
1		Организационная единица (Organizational unit)	Элемент организационной структуры (структурное подразделение), который отвечает за выполнение определенных задач и преследует определенные цели
2		Позиция или должность (Position)	Наиболее мелкий организационный элемент на предприятии. Обязанности и административные полномочия такого элемента задаются должностными инструкциями
3		Группа (Group)	Несколько сотрудников, которые вместе работают над конкретной задачей в определенный период времени (например, группа проектирования)
4		Внешний сотрудник (External person)	Лицо, выполняющее определенные функции в компании, но не являющееся служащим этой компании. Он может назначаться к организационным единицам или функциям, к выполнению которых он привлекается со стороны или за которые он отвечает
5		Сотрудник (Internal person)	Является служащим компании и может назначаться к организационным единицам или функциям, которые он выполняет или за которые несет ответственность
6		Тип сотрудника (Person type)	Обобщает индивидуальных лиц, имеющих одинаковые характеристики. Эти характеристики могут иметь отношение, например, к одним и тем же полномочиям. Начальники отделов или бригадиры, например, должны следовать определенным правилам и выполнять определенные обязанности, которые достаточно описать только один раз
7		Местонахождение (Location)	Определяет физическое положение (местонахождение) организационных единиц, рабочих мест, образцов аппаратных компонентов и технических ресурсов компании. Местонахождение может ссылаться на область, город, предприятие, здание и т. д.

При построении модели организационной структуры между структурными элементами могут устанавливаться следующие типы связей: имеет в подчинении (Is superior), связан с (Is assigned to), принадлежит (Belongs to), является организационным управляющим (Is Organization Manager for), имеет в своем составе (Has member), состоит из (Is composed of), располагает (Is located at), взаимодействует с (Cooperates with), содержит (Subsumes), занимает (Occupies).

Пример организационной диаграммы приведен на рисунке 2.4.

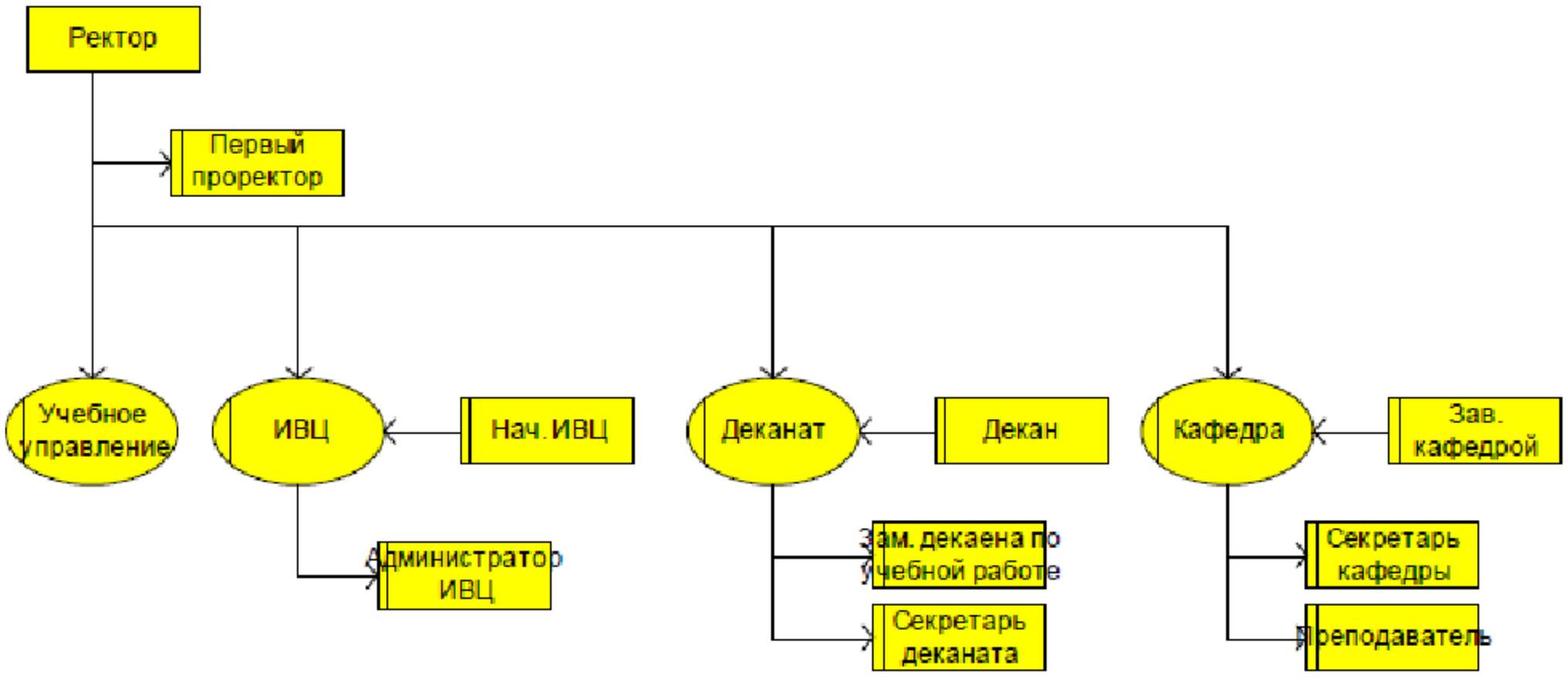


Рисунок 2.4 - Пример организационной диаграммы

2.3.2 Диаграмма цепочки добавленного качества

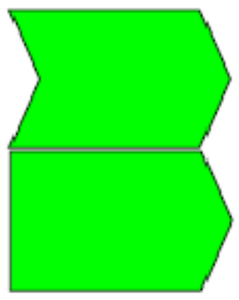
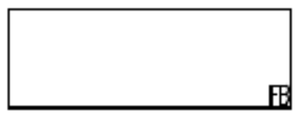
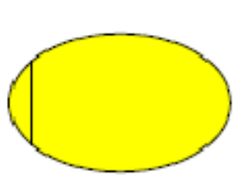
Для уменьшения сложности описания деятельности организации необходимо разработать иерархию моделей бизнес-процессов (БП) организации начиная с самого верхнего уровня и до моделей отдельных БП на нижнем уровне. Для описания модели верхнего уровня используется диаграмма типа Value-added chain diagram (VAD), название которой можно перевести как Модель цепочки добавленного качества (стоимости).

Диаграмма цепочек добавленного качества описывает функции организации, которые непосредственно влияют на реальный выход ее продукции. Эти функции создают последовательность действий, формируя добавленные значения: стоимость, количество, качество и т. д.

Диаграмма цепочек добавленной стоимости может содержать следующие структурные элементы: организационную единицу, должность или позицию, функцию и др. (таблица 2.7).

Пример диаграммы цепочек добавленной стоимости представлен на рисунок 2.5. Маленьким значком в нижнем правом углу элемента модели помечаются те функции, для которых существуют модели, раскрывающие эту функцию. Таким механизмом реализуется принцип декомпозиции.

Таблица 2.7 - Элементы диаграммы цепочки добавленного качества

№ п/п	Графическая нотация	Наименование	Описание
1		Функция	Действие или набор действий, выполняемых над исходным объектом (документом, материалом и т. п.) с целью получения заданного результата
2		Технический термин	
3		Организационная единица (Organizational unit)	Элемент организационной структуры (структурное подразделение), который отвечает за выполнение определенных задач и преследует определенные цели
4		Кластер (экземпляр)	Экземпляр объекта данных. Он представляет собой логический взгляд на набор объектов данных или структур

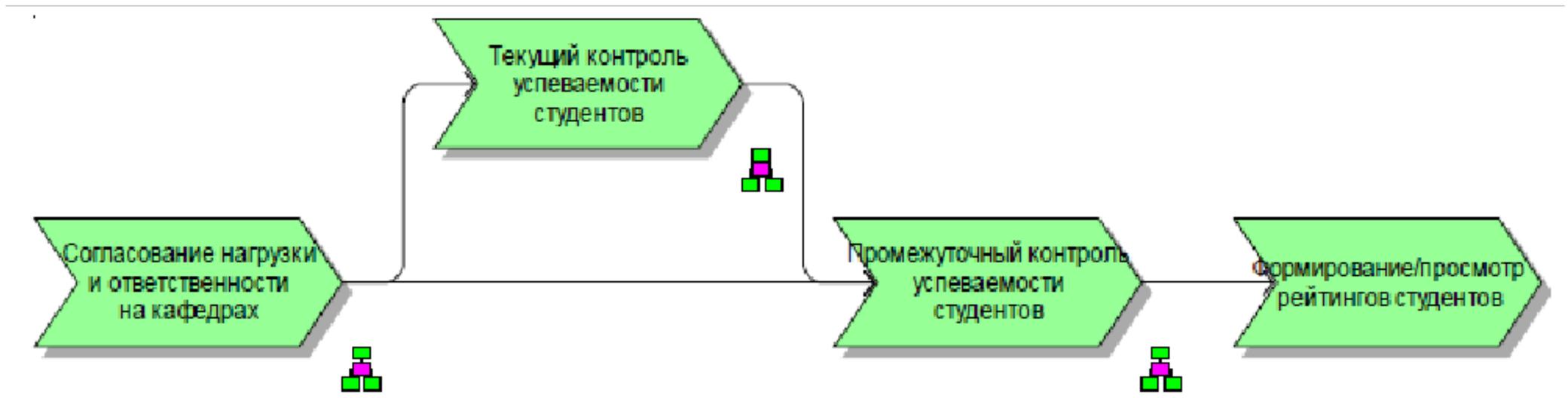


Рисунок 2.5 - Основные процессы учета успеваемости студентов в системе кредитов

2.3.3 Модели eEPC

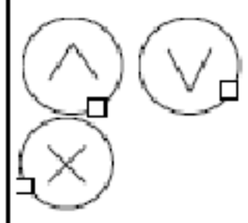
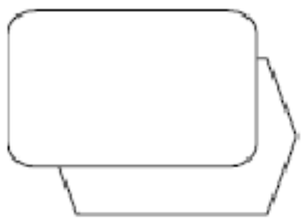
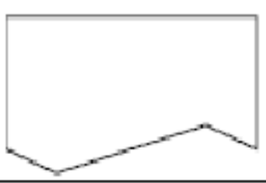
Цепочка процесса, управляемая событиями, – Extended event driven process chain (eEPC).

С помощью диаграмм eEPC процедуры БП представляются как логические последовательности событий. События определяют, какое состояние или отношение будет переключать функцию и какое состояние наступит после ее выполнения. Поэтому модель eEPC должна всегда иметь запускающие и завершающие события.

Одно событие может инициировать выполнение одновременно нескольких функций, и, наоборот, функция может быть результатом наступления нескольких событий. Объединения нескольких событий или функций отображаются на диаграмме eEPC с помощью соединителей в виде небольшого кружка. Эти соединители не только отображают графические связи между объектами модели, но и определяют логические связи между ними. Различают два типа связи логических операторов – **связи событий и связи функций**.

Диаграмма цепочек, управляемых событиями, кроме элементов организационной диаграммы и диаграммы данных, может содержать следующие структурные элементы: функцию, событие и др. (таблица 2.8).

Таблица 2.8 - Элементы диаграммы цепочек, управляемых событиями

№ п/п	Графическая нотация	Наименование	Описание
1		Функция	Действие или набор действий, выполняемых над исходным объектом (документом, материалом и т. п.) с целью получения заданного результата
2		Событие	Состояние, которое является существенным для управления БП и которое оказывает влияние или контролирует дальнейшее развитие одного или более БП. Изменения состояния отражаются с помощью информационных объектов
3		Правило	Представляет собой правило разветвления и слияния веток БП. Если перейти к рассмотрению каждой отдельной функции БП, то можно сказать, что правило отражает логическое соотношение между несколькими исходными для функции событиями и несколькими результирующими
4		Интерфейс функции	Используется для обозначения функции внешнего БП, который либо не описывается в данной модели, либо является БП другой предметной области
5		Носитель информации	Представляет собой средство хранения информации. Оно может быть реализовано, к примеру, в виде картотеки или компьютерных файлов или БД
6		Документ	Обозначает любой вид документов

Пример диаграммы цепочек, управляемых событиями, представлен на рисунок 2.6.

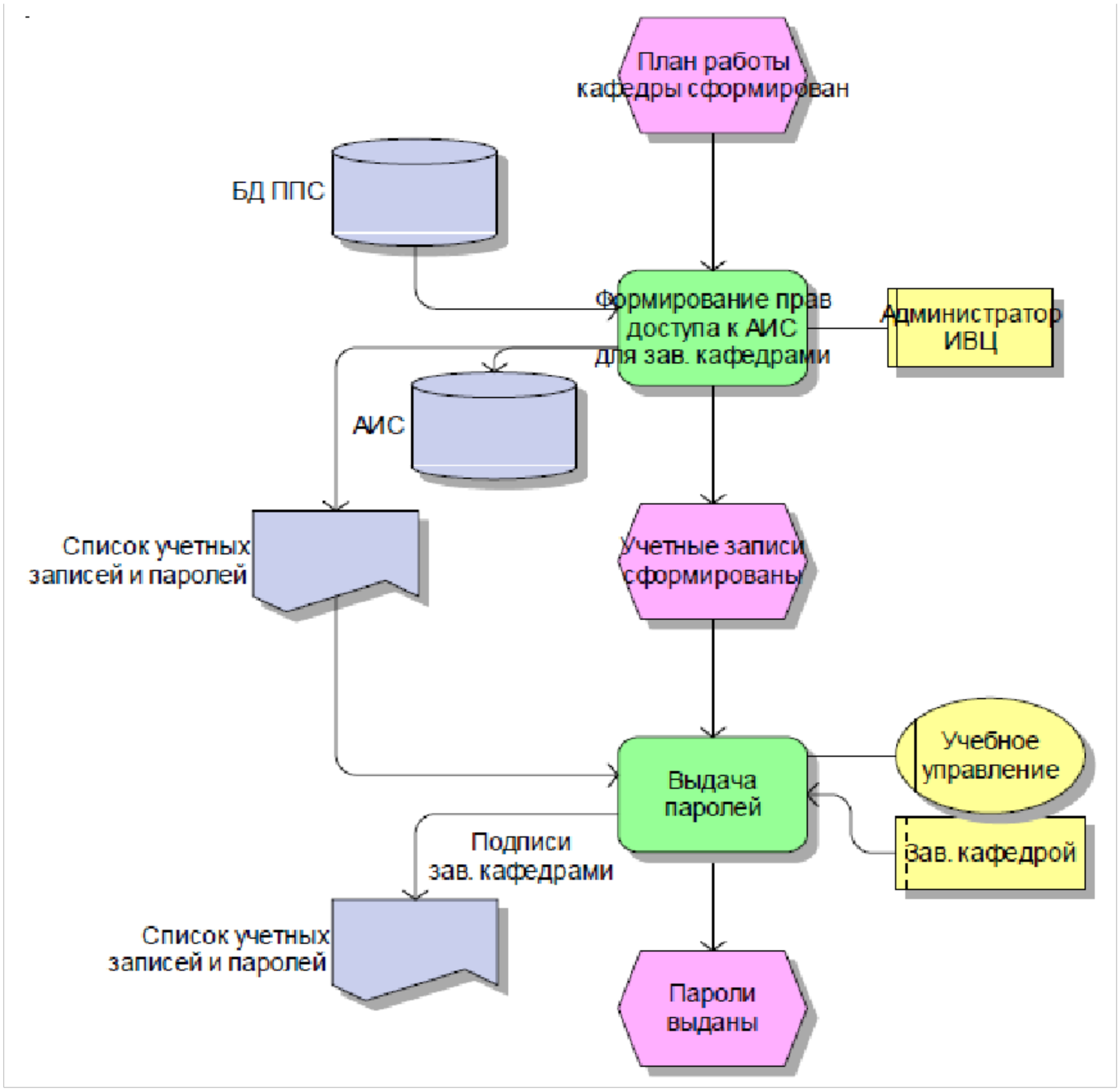


Рисунок 2.6 - Согласование нагрузки и ответственности на кафедрах

2.4 Стандарты IDEF0–IDEF3

2.4.1 Методология описания бизнес-процессов IDEF3

IDEF3 – способ описания процессов, основной целью которого является обеспечение структурированного метода, с помощью которого эксперт в предметной области может описать положение вещей как упорядоченную последовательность событий с одновременным описанием объектов, имею- щих непосредственное отношение к процессу.

Технология IDEF3 хорошо приспособлена для сбора данных, требующихся для проведения структурного анализа системы. В отличие от большинства технологий моделирования бизнес-процессов, IDEF3 не имеет жестких синтаксических или семантических ограничений, делающих неудобным описание неполных или нецелостных систем. Кроме того, автор модели (системный аналитик) избавлен от необходимости смешивать свои собственные предположения о функционировании системы с экспертными утверждениями в целях заполнения пробелов в описании предметной области. На рисунок 2.7 изображен пример описания процесса с использованием методологии IDEF3.

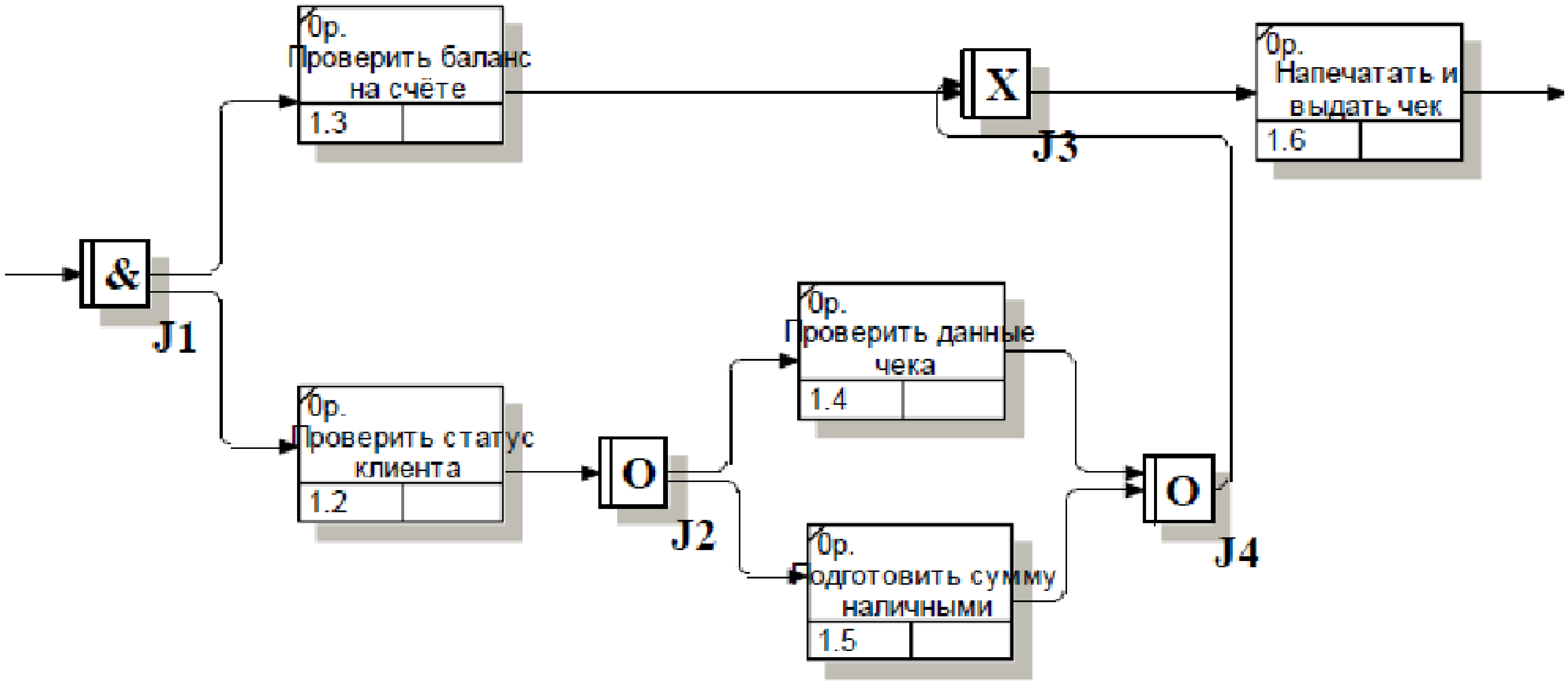


Рисунок 2.7 - Описание процесса в методологии IDEF3

Технология IDEF3 также может быть использована как метод проектирования бизнес-процессов. IDEF3-моделирование органично дополняет традиционное моделирование с использованием стандарта IDEF0. В настоящее время оно получает все большее распространение как вполне жизнеспособный путь построения моделей проектируемых систем для дальнейшего анализа имитационными методами. Имитационное тестирование часто используют для оценки эксплуатационных качеств разрабатываемой системы, более подробно методы имитационного анализа будут рассмотрены позднее.

2.4.1.1 Синтаксис и семантика моделей IDEF

Модели IDEF3. Основой модели IDEF3 служит так называемый сценарий бизнес-процесса, который выделяет последовательность действий или подпроцессов анализируемой системы. Поскольку сценарий определяет назначение и границы модели, довольно важным является подбор подходящего наименования для обозначения действий. Для подбора необходимого имени применяются стандартные рекомендации по предпочтительному использованию глаголов и отглагольных существительных. Например, «Обработать заказ клиента» или «Применить новый дизайн» – вполне подходящие названия сценариев.

Точка зрения для большинства моделей должна быть явным образом документирована. Обычно это название набора должностных обязанностей человека, являющегося источником информации о моделируемом процессе.

Для системного аналитика также важно понимание цели моделирования – набора вопросов, ответами на которые будет служить модель, границ моделирования (какие части системы войдут в модель, а какие не будут в ней отображены) и целевой аудитории (для кого разрабатывается модель).

Диаграммы. Главной организационной единицей модели IDEF3 является диаграмма. Взаимная организация диаграмм внутри модели IDEF3 особенно важна в случае, когда модель заведомо создается для последующего опубликования или рецензирования, что является вполне обычной практикой при проектировании новых систем. В этом случае системный аналитик должен позаботиться о таком информационном наполнении диаграмм, чтобы каждая из них была самодостаточной и в то же время понятной читателю.

Единица работы. Действие. Аналогично другим технологиям моделирования действие, или в терминах IDEF3 «единица работы» (Unit of Work – UOW), – другой важный компонент модели. Диаграммы IDEF3 отображают действие в виде прямоугольника. Как уже отмечалось, действия именуются с использованием глаголов или отглагольных существительных, каждому из действий присваивается уникальный идентификационный номер. Этот номер не используется вновь даже в том случае, если в процессе построения модели действие удаляется. В

диаграммах IDEF3 номер действия обычно предваряется номером его родителя (рисунок 2.8).

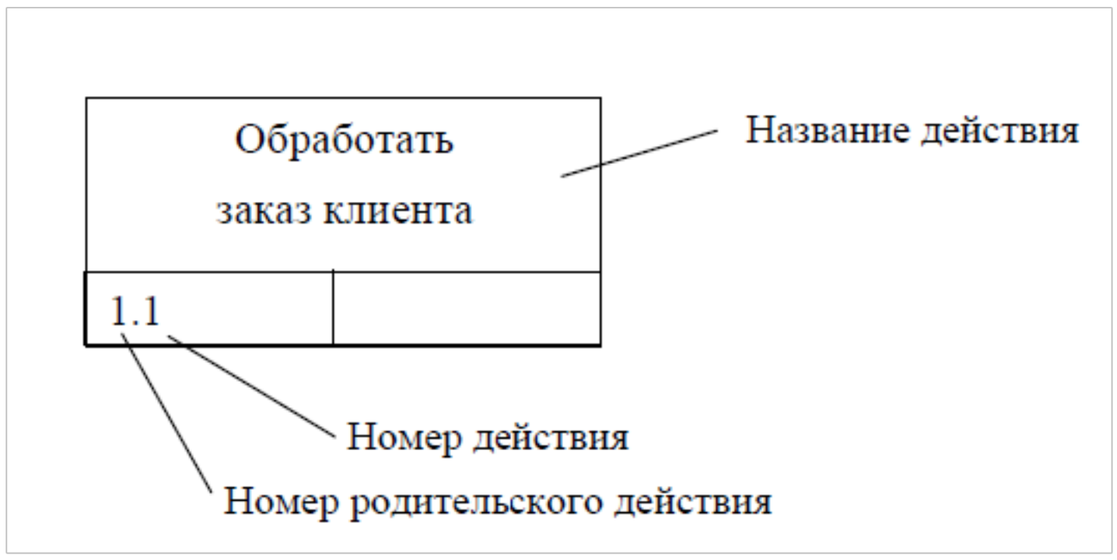


Рисунок 2.8 - Изображение и нумерация действия в диаграмме IDEF3

Связи. Связи выделяют существенные взаимоотношения между действиями. Все связи в IDEF3 являются однонаправленными, и, хотя стрелка может начинаться или заканчиваться на любой стороне блока, обозначающего действие, диаграммы IDEF3 обычно организовываются слева направо таким образом, что стрелки начинаются на правой и заканчиваются на левой стороне блоков. В таблице 2.9 приведены три возможных типа связей.

Таблица 2.9 - Типы связей в модели IDEF3

Изображение	Название	Назначение
→	Временное предшествование (Temporal precedence)	Исходное действие должно завершиться прежде, чем конечное действие сможет начаться
→→	Объектный поток (Object flow)	Выход исходного действия является входом конечного действия. Из этого, в частности, следует, что исходное действие должно завершиться прежде, чем конечное действие сможет начаться
- - - →	Нечеткое отношение (Relationship)	Вид взаимодействия между исходным и конечным действиями задается аналитиком отдельно для каждого случая использования такого отношения

Связь типа «временное предшествование». Как видно из названия, связи этого типа отражают, что исходное действие должно полностью завершиться, прежде чем начнется выполнение конечного действия. Связь должна быть поименована таким образом, чтобы человеку, просматривающему модель, была понятна причина ее появления. Во многих случаях завершение одного действия инициирует начало выполнения другого (рисунок 2.9). В этом примере автор должен принять рекомендации рецензентов, прежде чем начать вносить соответствующие изменения в работу.

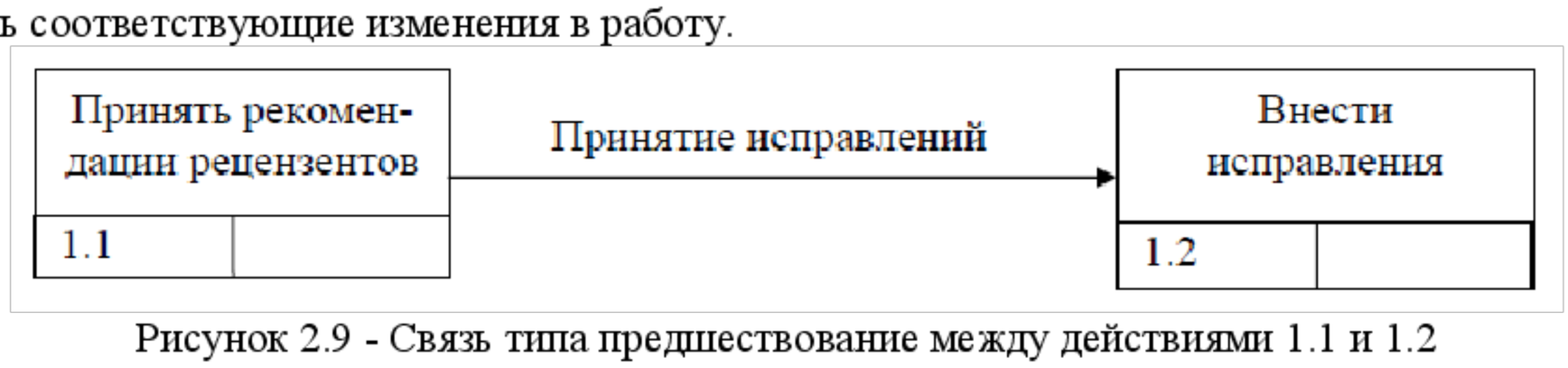


Рисунок 2.9 - Связь типа предшествование между действиями 1.1 и 1.2

Связь типа «объектный поток». Одной из наиболее часто встречающихся причин использования связи типа «объектный поток» состоит в том, что некоторый объект, являющийся результатом выполнения исходного действия, необходим для выполнения конечного действия. Такая связь отличается от связи временного предшествования двойным концом обозначающей ее стрелки. Наименования потоковых связей должны четко идентифицировать объект, который передается с их помощью. Временная семантика объектных связей аналогична связям предшествования. Это означает, что порождающее объектную связь исходное действие должно завершиться, прежде чем конечное действие начнет выполняться (рисунок 2.10). В приведенном примере счет на оплату услуг является результатом выполнения действия 1.1. Счет

необходим для проведения оплаты услуг.

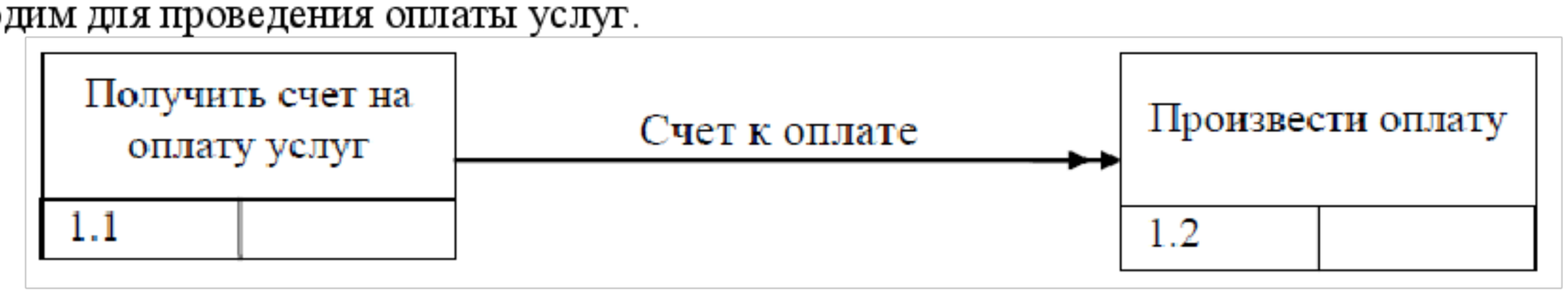


Рисунок 2.10 - Объектная связь между действиями 1.1 и 1.2

Связь типа «нечеткое отношение». Связи этого типа используются для выделения отношений между действиями, которые невозможно описать с использованием предшественных или объектных связей. Значение каждой такой связи должно быть определено, поскольку связи типа нечеткое отношение сами по себе не предполагают никаких ограничений. Одно из применений нечетких отношений – отображение взаимоотношений между параллельно выполняющимися действиями. Рисунок 2.11 иллюстрирует фрагмент процесса запуска бензопилы с водяным охлаждением и нечеткое отношение между действиями запустить двигатель и запустить водяной насос. Название стрелки может быть использовано для описания природы отношения, более подробное объяснение может быть приведено в виде отдельной ссылки.

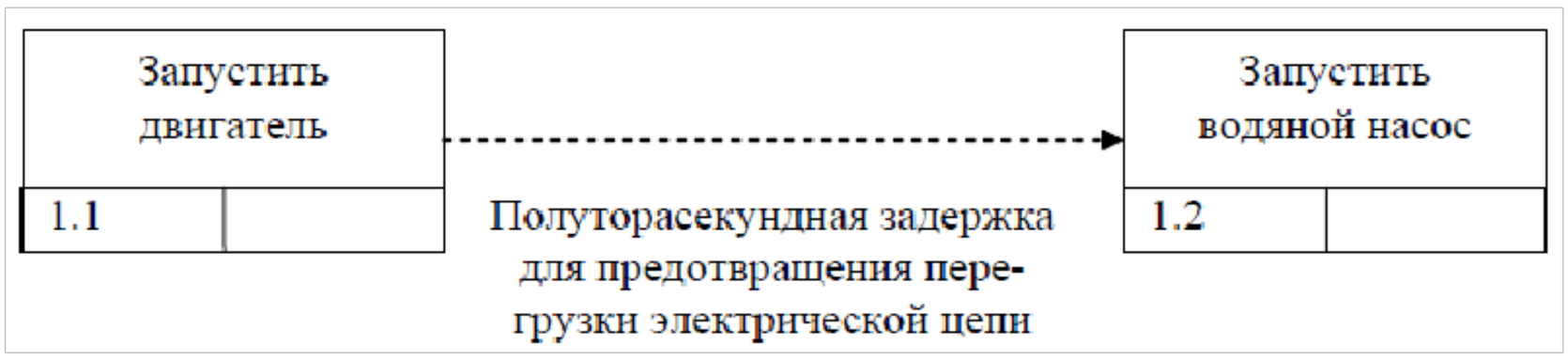


Рисунок 2.11 - Связь типа «нечеткое отношение»

Наиболее часто нечеткие отношения используются для описания специальных случаев связей предшествования, например для описания альтернативных вариантов временного предшествования. Альтернативная предшественной связи (рисунок 2.9) связь нечеткого отношения представлена на рисунок 2.12. В этом примере внесение исправлений начинается по мере получения замечаний от рецензентов, т. е. до непосредственного окончания действия по принятию замечаний.

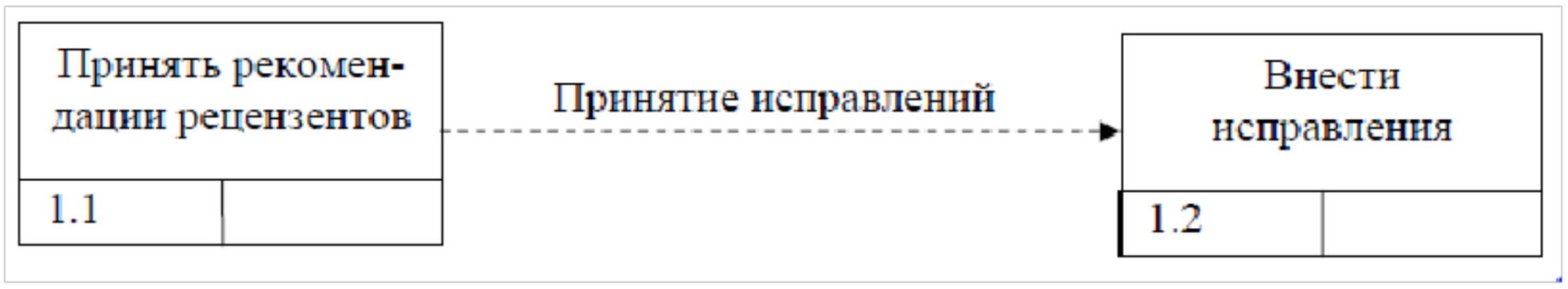


Рисунок 2.12 - Альтернатива связи предшествования

Соединения. Завершение одного действия может инициировать начало выполнения сразу нескольких других действий, или, наоборот, определенное действие может требовать завершения нескольких других действий для начала своего выполнения. Соединения разбивают или соединяют внутренние потоки и используют для описания ветвления процесса.

Разворачивающие соединения используются для разбиения потока. Завершение одного действия вызывает начало выполнения нескольких других.

Сворачивающие соединения объединяют потоки. Завершение одного или нескольких действий вызывает начало выполнения только одного другого действия. В таблице 2.10 приведены три типа соединений.

Таблица 2.10 - Типы соединений в модели IDEF3

Графическое обозначение	Название	Вид	Правила инициации
&	Соединение «И»	Разворачивающее	Каждое конечное действие обязательно инициируется
		Сворачивающее	Каждое исходное действие обязательно должно завер-
X	Соединение «Исключающее ИЛИ»	Разворачивающее	Одно и только одно конечное действие инициируется
		Сворачивающее	Одно и только одно исходное действие должно за-
O	Соединение «ИЛИ»	Разворачивающее	Одно (или более) конечное действие инициируется
		Сворачивающее	Одно (или более) исходное действие должно завер-

Примеры разворачивающих и сворачивающих соединений приведены на рисунке 2.13.

«И»-соединения. Соединения этого типа инициируют выполнение всех своих конечных действий. Все действия, присоединенные к сворачивающему «И»-соединению, должны завершиться, прежде чем может начать выполняться следующее действие. После обнаружения пожара инициируются включение пожарной сигнализации (рисунок 2.14), вызов пожарной охраны и начинается тушение пожара. Запись в журнал производится только тогда, когда все три перечисленных действия завершены.

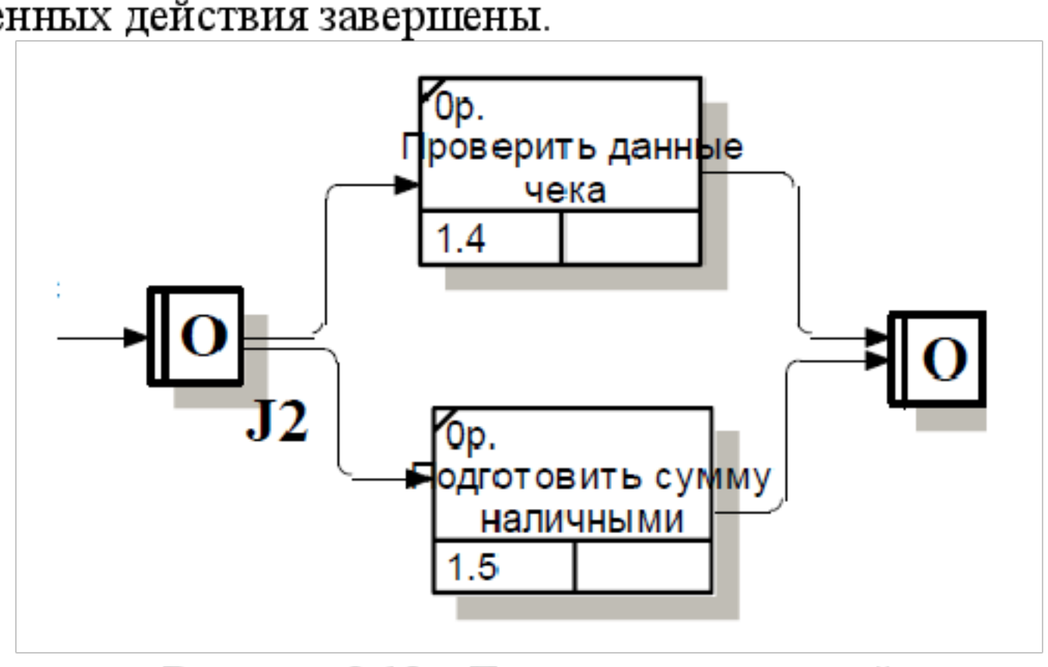


Рисунок 2.13 - Два вида соединений

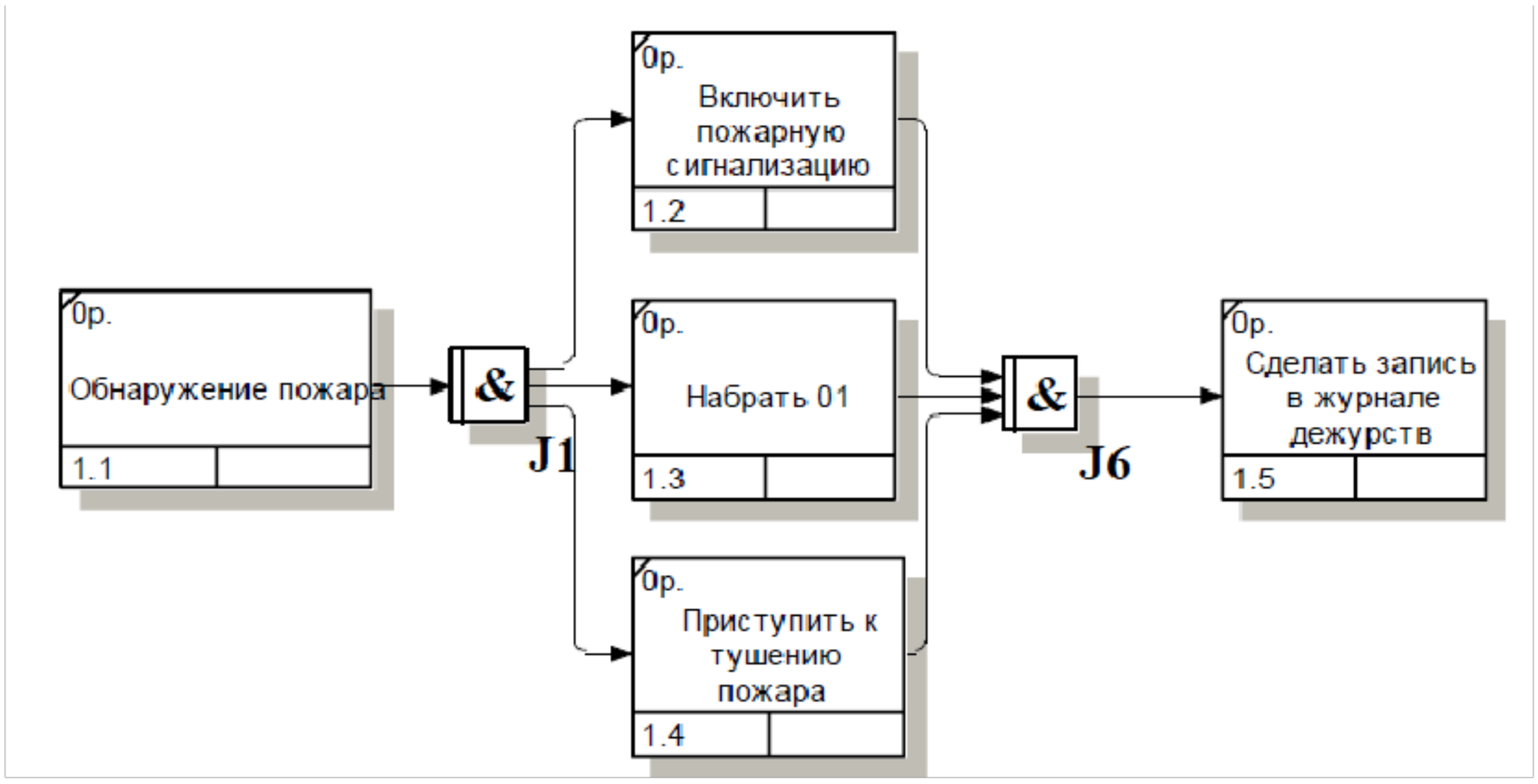


Рисунок 2.14 - «И»-соединения

Соединение «Исключающее ИЛИ». Вне зависимости от количества действий, прицепленных к сворачивающему или разворачивающему соединению «Исключающее ИЛИ», инициировано будет только одно из них, и поэтому только одно из них будет завершено перед тем, как любое действие, следующее за сворачивающим соединением «Исключающее ИЛИ», сможет начаться.

Если правила активации соединения известны, они обязательно должны быть документированы либо в его описании, либо пометкой стрелок, исходящих из разворачивающего соединения (рисунок 2.15).

Соединение «Исключающее ИЛИ» используется для отображения того факта, что студент не может одновременно быть направлен на лекции по двум разным курсам.

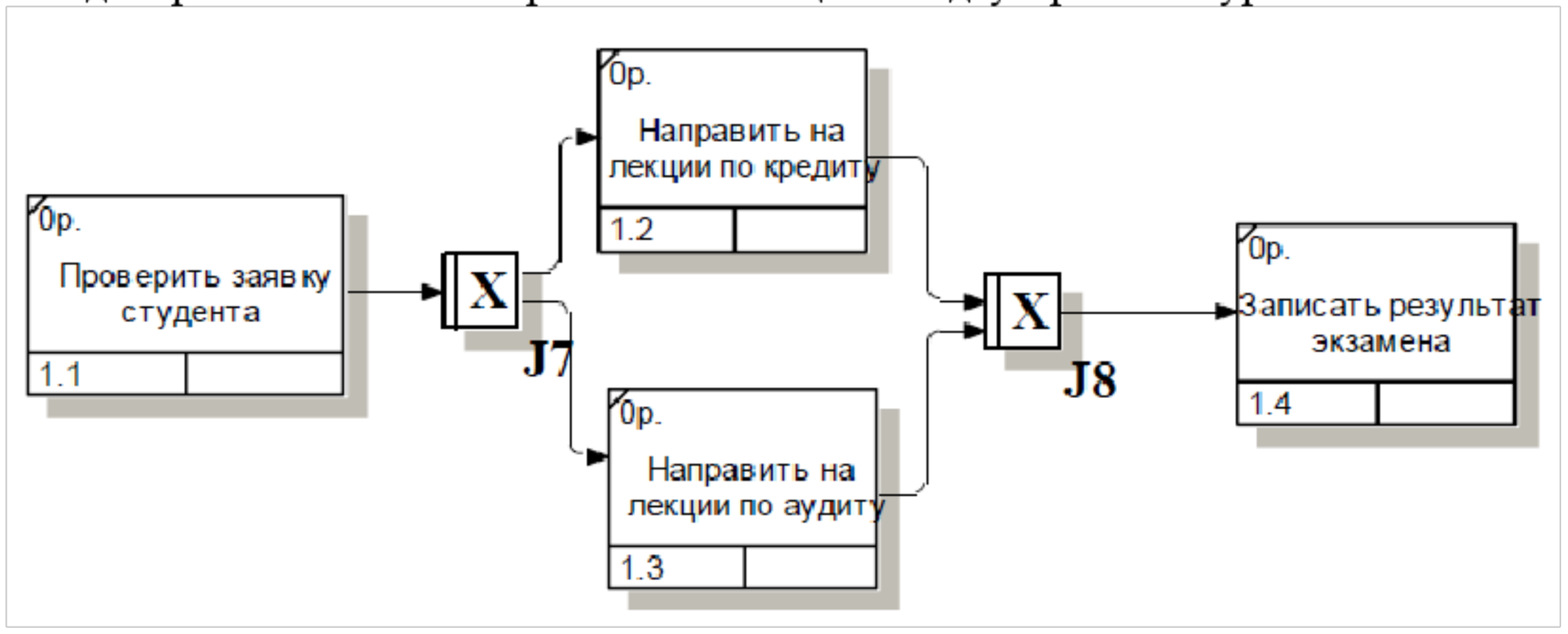


Рисунок 2.15 - Соединение «Исключающее ИЛИ»

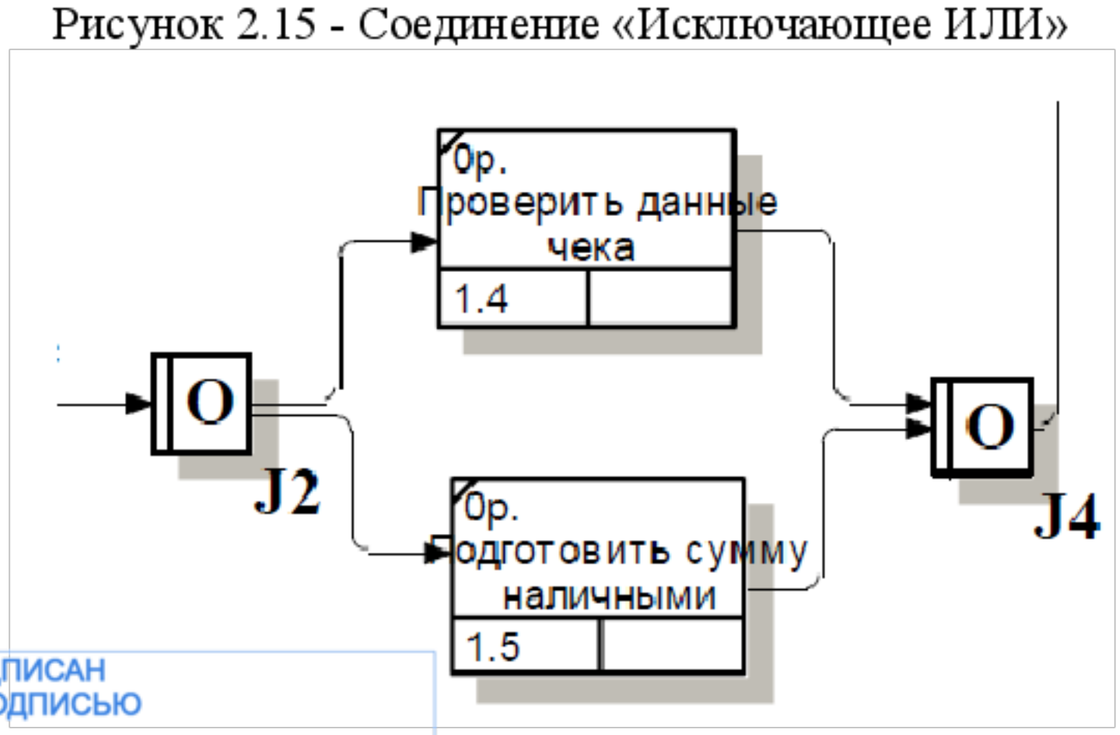


Рисунок 2.16 - Соединение «ИЛИ»

Соединение «ИЛИ». Соединения этого типа предназначены для описания ситуаций,

которые не могут быть описаны двумя предыдущими типами соединений. Аналогично связи нечеткого отношения соединение «ИЛИ» в основном определяется и описывается непосредственно системным аналитиком. Соединение J2 может активировать проверку данных чека и (или) проверку суммы наличных (рисунок 2.16). Проверка чека инициируется, если покупатель желает расплатиться чеком, проверка суммы наличных – при оплате наличными. Оба действия инициируются при частичной оплате чеком и частичной – наличными.

Декомпозиция действий. Действия в IDEF3 могут быть декомпозированы или разложены на составляющие, для более детального анализа. Декомпозировать действие можно несколько раз. Это позволяет документировать альтернативные потоки процесса в одной модели.

Для корректной идентификации действий в модели с множественными декомпозициями схема нумерации действий расширяется и наряду с номерами действия и его родителя включает в себя порядковый номер декомпозиции. Например, в номере действия 1.2.5: 1 – номер родительского действия, 2 – номер декомпозиции, 5 – номер действия.

2.4.1.2 Требования IDEF3 к описанию бизнес-процессов

Здесь мы рассмотрим построение IDEF3-диаграммы на основе выраженного в текстовом виде описания процесса. Предполагается, что в построении диаграммы принимают участие ее автор (в основном как системный аналитик) и один или несколько экспертов предметной области, которые будут представлять нам описание процесса.

Определение сценария, границ моделирования, точки зрения. Перед тем как попросить экспертов предметной области подготовить описание моделируемого процесса, должны быть документированы границы моделирования, чтобы экспертам была понятна необходимая глубина и полнота требуемого от них описания. Кроме того, если точка зрения аналитика на процесс отличается от обычной точки зрения для эксперта, это должно быть ясно и аккуратно описано.

Вполне возможно, что эксперты не смогут сделать приемлемое описание без применения формального опроса автором модели. В таком случае автор должен заранее приготовить набор вопросов таким же образом, как журналист заранее подготавливает вопросы для интервью.

Определение действий и объектов. Результатом работы экспертов обычно является текстовый документ, описывающий интересующий аналитика круг вопросов. В дополнение к нему может иметься письменная документация, позволяющая пролить свет на природу изучаемого процесса. Вне зависимости от того, является ли информация текстовой или вербальной, она анализируется и разделяется частями речи для идентификации списка действий (глаголы и отглагольные существительные), составляющих процесс, и объектов (имена существительные), участвующих в процессе.

В некоторых случаях возможно создание графической модели процесса в присутствии экспертов. Такая модель также может быть разработана после сбора всей необходимой информации, что позволяет не отнимать время экспертов на детали форматирования получающихся диаграмм.

Поскольку модели IDEF3 могут одновременно разрабатываться несколькими командами, IDEF3 поддерживает простую схему резервирования номеров действий в модели. Каждому аналитику выделяется уникальный диапазон номеров действий, что обеспечивает их независимость друг от друга. В таблице 2.11 номера действий выделяются каждому аналитику большими блоками. В этом примере Иван исчерпал данный ему вначале диапазон номеров и дополнительно получил второй.

Таблица 2.11 - Распределение диапазонов номеров IDEF3 между аналитиками

Аналитик	Диапазон номеров IDEF3
Иван	1–99
Петр	100–199
Николай	200–299
Иван	300–399

Последовательность и параллельность. Если модель создается после проведения

интервью, аналитик должен принять решения по построению иерархии участвующих в модели диаграмм, например, насколько подробно будет детализироваться каждая отдельно взятая диаграмма. Если последовательность или параллельность выполнения действий окончательно не ясна, эксперты могут быть опрошены вторично (возможно, с использованием черновых вариантов незаконченных диаграмм) для получения недостающей информации. Важно, однако, различать предполагаемую (появляющуюся из- за недостатка информации о связях) и явную (ясно указанную в описании эксперта) параллельности.

Итак, IDEF3 – это способ описания бизнес-процессов, который нужен для описания положения вещей как упорядоченной последовательности событий с одновременным описанием объектов, имеющих непосредственное отношение к процессу. IDEF3 хорошо приспособлен для сбора данных, требующихся для проведения структурного анализа системы. Кроме того, IDEF3 применяется при проведении стоимостного анализа поведения моделируемой системы.

2.4.2 Методология функционального моделирования IDEF0

Методология функционального моделирования IDEF0 – это технология описания системы в целом как множества взаимозависимых действий, или функций. Важно отметить функциональную направленность IDEF0 – функции системы исследуются независимо от объектов, которые обеспечивают их выполнение. «Функциональная» точка зрения позволяет четко отделить аспекты назначения системы от аспектов ее физической реализации. На рисунке 2.17 приведен пример типовой диаграммы IDEF0.

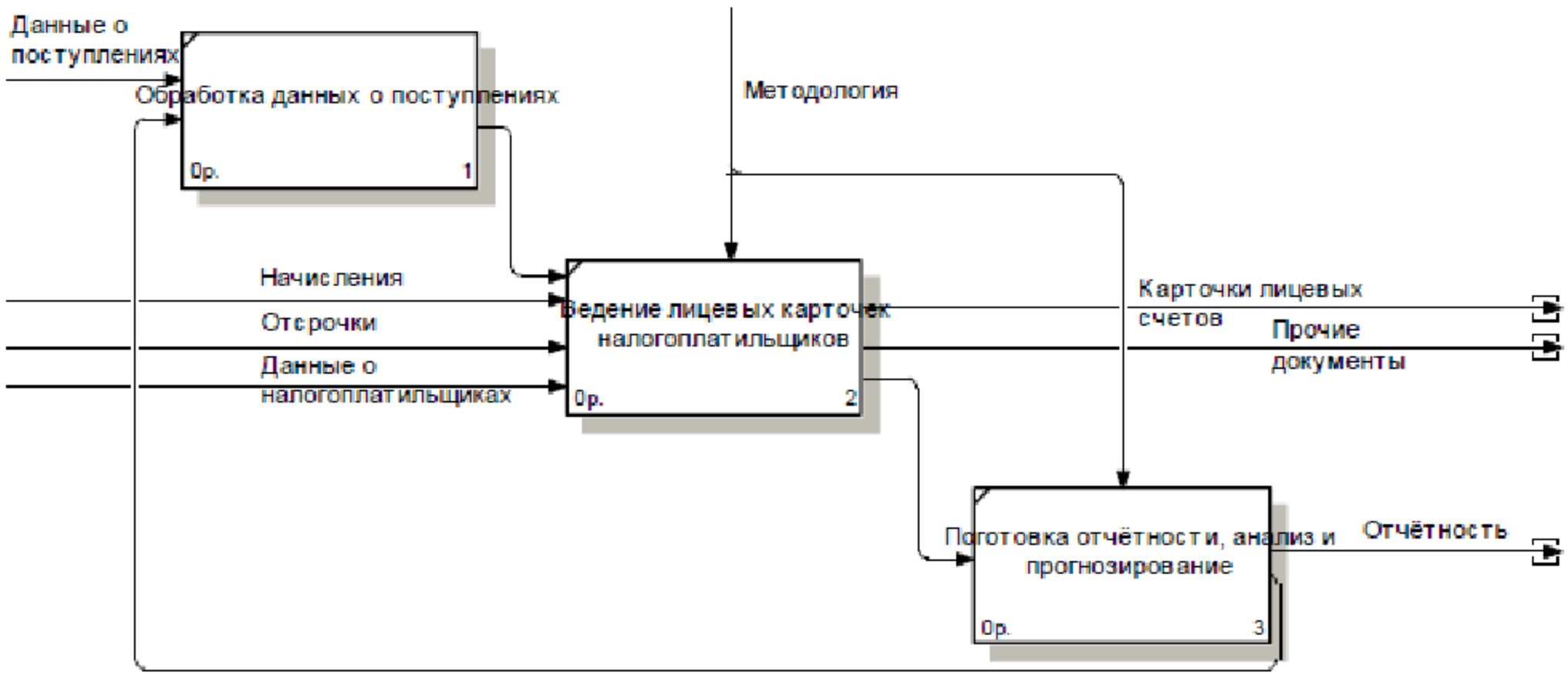


Рисунок 2.17 - Пример диаграммы IDEF0

Наиболее часто IDEF0 применяется как технология исследования и проектирования систем на логическом уровне. По этой причине он, как правило, используется на ранних этапах разработки проекта, до IDEF3-моделирования для сбора данных и моделирования процесса «как есть». Результаты IDEF0-анализа могут применяться при проведении проектирования с использованием моделей IDEF3 и диаграмм потоков данных.

2.4.2.1 Синтаксис и семантика моделей IDEF0

Модели IDEF0. IDEF0 сочетает в себе небольшую по объему графическую нотацию (она содержит только два обозначения: блоки и стрелки) со строгими и четко определенными рекомендациями, в совокупности предназначенными для построения качественной и понятной модели системы.

Методология IDEF0 в некоторой степени напоминает рекомендации, существующие в книгоиздательском деле, часто набор напечатанных моделей IDEF0 организуется в брошюру (называемую в терминах IDEF0 комплект), имеющую содержание, глоссарий и другие элементы, характерные для законченной книги.

Первый шаг при построении модели IDEF0 заключается в определении назначения модели – набора вопросов, на которые должна отвечать модель.

Набор вопросов можно сравнить с предисловием, в котором раскрывается назначение книги.

Границы моделирования предназначены для обозначения ширины охвата предметной области и глубины детализации и являются логическим продолжением уже определенного назначения модели. Как читающий модель, так и непосредственно ее автор должны понимать степень детальности ответов на поставленные в назначении модели вопросы.

Следующим шагом указывается предполагаемая целевая аудитория, для нужд которой создается модель. Зачастую от выбора целевой аудитории зависит уровень детализации, с которым должна создаваться модель. Перед построением модели необходимо иметь представление о том, какие сведения о предмете моделирования уже известны, какие дополнительные материалы и (или) техническая документация для понимания модели могут быть необходимы целевой аудитории, какие язык и стиль изложения являются наиболее подходящими.

Под точкой зрения понимается перспектива, с которой наблюдалась система при построении модели. Точка зрения выбирается таким образом, чтобы учесть уже обозначенные границы моделирования и назначение модели. Однажды выбранная точка зрения остается неизменной для всех элементов модели. При необходимости могут быть созданы другие модели, отображающие систему с других точек зрения. Вот несколько примеров точек зрения при построении моделей: клиент, поставщик, владелец, редактор.

Действия. Действие, обычно в IDEF0 называемое функцией, обрабатывает или переводит входные параметры (сырье, информацию и т. п.) в выходные. Поскольку модели IDEF0 представляют систему как множество иерархических (вложенных) функций, в первую очередь должна быть определена функция, описывающая систему в целом – контекстная функция. Функции изображаются на диаграммах как поименованные прямоугольники, или функциональные блоки. Имена функций в IDEF0 подбираются по сходным правилам с именами действий в IDEF3 – с использованием глаголов или отглагольных существительных. Важно подбирать имена таким образом, чтобы они отражали систему так, как если бы она обозревалась с точки зрения, выбранной для моделирования.

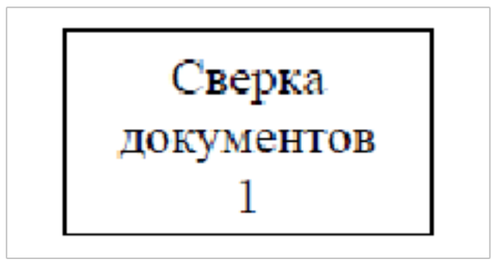


Рисунок 2.18 - Функциональный блок IDEF0

Пример функционального блока приведен на рисунке 2.18.

Выше мы определяли IDEF0-модели как иерархическое множество вложенных блоков. Любой блок может быть декомпозирован на составляющие его блоки. Декомпозицию часто ассоциируют с моделированием «сверху вниз», однако это не совсем верно. Функциональную декомпозицию корректнее определять как моделирование «снаружи вовнутрь», в котором мы рассматриваем систему наподобие луковицы, с которой последовательно снимаются слои.

Границы и связи. Чтобы быть полезным, описание любого блока должно, как минимум, включать в себя описание объектов, которые блок создает в результате своей работы («выхода»), и объектов, которые блок потребляет или преобразует («вход»).

В IDEF0 также моделируются управление и механизмы исполнения. Под управлением понимаются объекты, воздействующие на способ, которым блок преобразует вход в выход. Механизм исполнения – объекты, которые непосредственно выполняют преобразование входа в выход, но не потребляются при этом сами по себе.

Для отображения категорий информации, присутствующих на диаграммах IDEF0, существует аббревиатура ICOM, отображающая четыре возможных типа стрелок:

I (Input) – вход – нечто, что потребляется в ходе выполнения процесса;

C (Control) – управление – ограничения и инструкции, влияющие на ход выполнения процесса;

O (Output) – выход – нечто, являющееся результатом выполнения процесса;

M (Mechanism) – исполняющий механизм – нечто, что используется для выполнения процесса, но не потребляется само по себе. Рисунок 2.19 показывает четыре возможных типа стрелок в IDEF0, каждый из типов соединяется со своей стороной функционального блока.

Для названия стрелок, как правило, употребляются имена существительные. Стрелки

могут представлять собой людей, места, вещи, идеи или события. Как и в случае с функциональными блоками, присвоение имен всем стрелкам на диаграмме является только необходимым условием для понимания читателем сути изображенного. Отдельное описание каждой стрелки в текстовом виде может оказаться критическим фактором для построения точной и полезной модели.

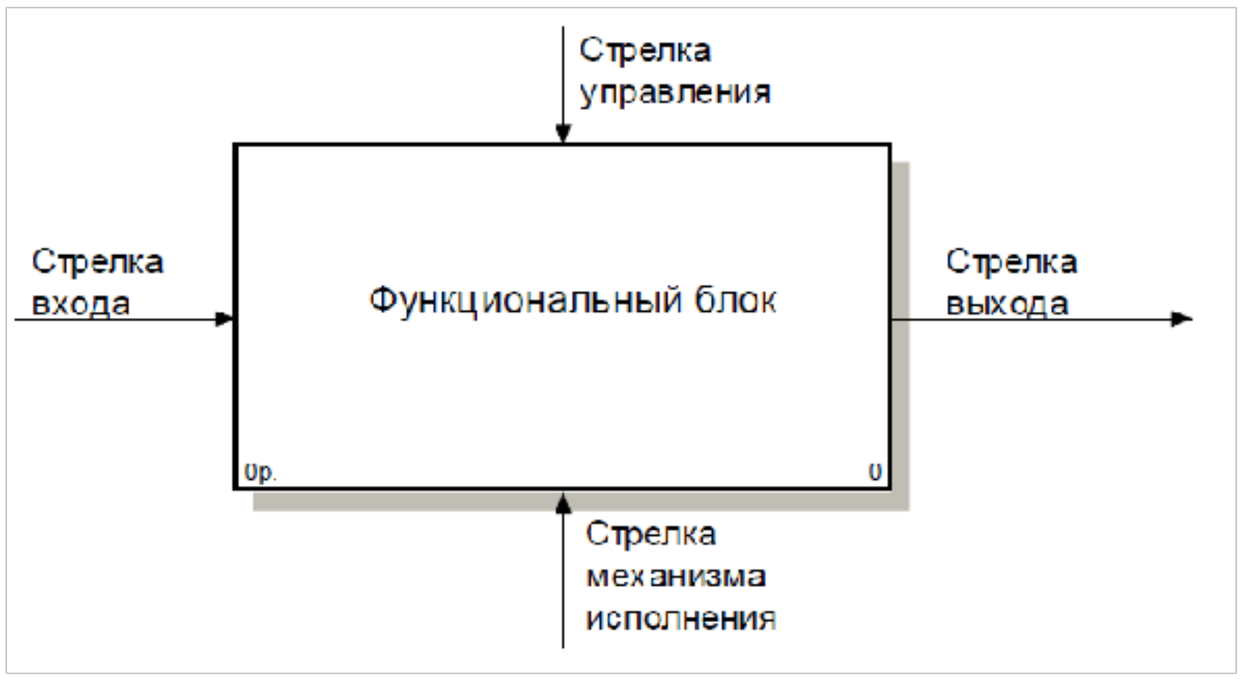


Рисунок 2.19 - Соединение стрелок со сторонами функционального блока

Стрелки входа. Вход представляет собой сырье, или информацию, потребляемую или преобразуемую функциональным блоком для производства выхода. Стрелки входа всегда направлены в левую сторону прямоугольника, обозначающего в IDEF0 функциональный блок. Наличие входных стрелок на диаграмме не является обязательным, так как возможно, что некоторые блоки ничего не преобразуют и не изменяют. Примером блока, не имеющего входа, может служить блок «Принятие решения руководством», где для принятия решения анализируется несколько факторов, но ни один из них непосредственно не преобразуется и не потребляется в результате принятия какого-либо решения.

Стрелки управления. Стрелки управления отвечают за регулирование того, как и когда выполняется функциональный блок, и, если он выполняется, какой выход получается в результате его выполнения. Так как управление контролирует поведение функционального блока для обеспечения создания желаемого выхода, каждый функциональный блок должен иметь, как минимум, одну стрелку управления. Стрелки управления всегда входят в функциональный блок сверху.

Управление часто существует в виде правил, инструкций, законов, политики, набора необходимых процедур или стандартов. Влияя на работу блока, оно непосредственно не потребляется и не трансформируется в результате. Может оказаться, что целью функционального блока является как раз изменение того или иного правила, инструкции, стандарта и т. п. В этом случае стрелка, содержащая соответствующую информацию, должна рассматриваться не как управление, а как вход функционального блока.

Управление можно рассматривать как специфический вид входа. В случаях, когда неясно, относить ли стрелку к входу или к управлению, предпочтительно относить ее к управлению до момента, пока неясность не будет разрешена.

Стрелки выхода. Выход – это продукция или информация, получаемая в результате работы функционального блока. Каждый блок должен иметь, как минимум, один выход. Действие, которое не производит никакого четко определяемого выхода, не должно моделироваться вообще (по меньшей мере, должно рассматриваться в качестве одного из первых кандидатов на исключение из модели).

При моделировании непроизводственных предметных областей выходами, как правило, являются данные, в каком-либо виде обрабатываемые функциональным блоком. В этом случае важно, чтобы названия стрелок входа и выхода были достаточно различимы по своему смыслу. Например, блок «Прием пациентов» может иметь стрелку «Данные о пациенте» как на входе, так и на выходе. В такой ситуации входящую стрелку можно назвать «Предварительные данные о пациенте», а исходящую – «Подтвержденные данные о пациенте».

Стрелки механизма исполнения. Механизмы являются ресурсом, который непосредственно исполняет моделируемое действие. С помощью механизмов исполнения могут моделироваться: ключевой персонал, техника и (или) оборудование. Стрелки механизма исполнения могут отсутствовать в случае, если оказывается, что они не являются

необходимыми для достижения поставленной цели моделирования.

Комбинированные стрелки. В IDEF0 существует пять основных видов комбинированных стрелок: «Выход-вход», «Выход – управление», «Выход – механизм исполнения», «Выход – обратная связь на управление» и «Выход – обратная связь на вход».

Стрелка «Выход-вход» применяется, когда один из блоков должен полностью завершить работу перед началом работы другого блока. Так, на рисунке 2.20 формирование счета должно предшествовать приему заказа.

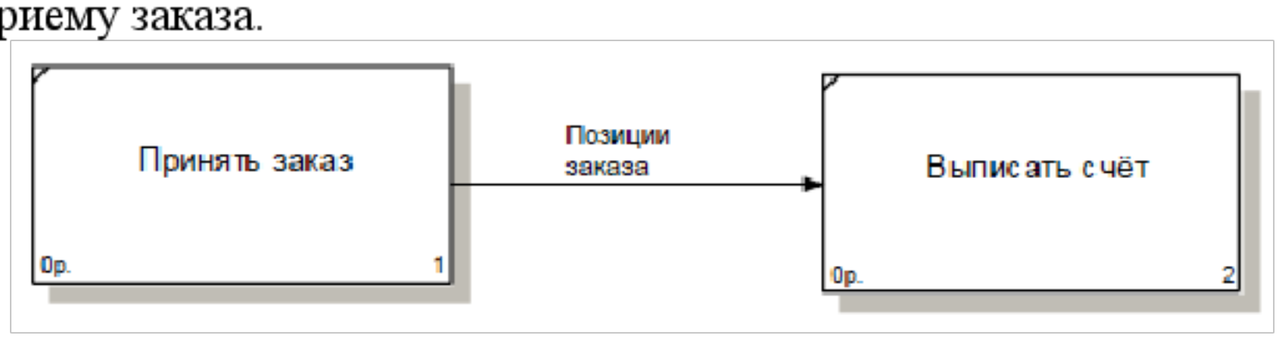


Рисунок 2.20 - Комбинация стрелок «Выход-вход»

Стрелка «Выход – управление» отражает ситуацию преобладания одно- го блока над другим, когда один блок управляет работой другого. На рисунке 2.21 принципы формирования инвестиционного портфеля управляют поведением брокеров на бирже.

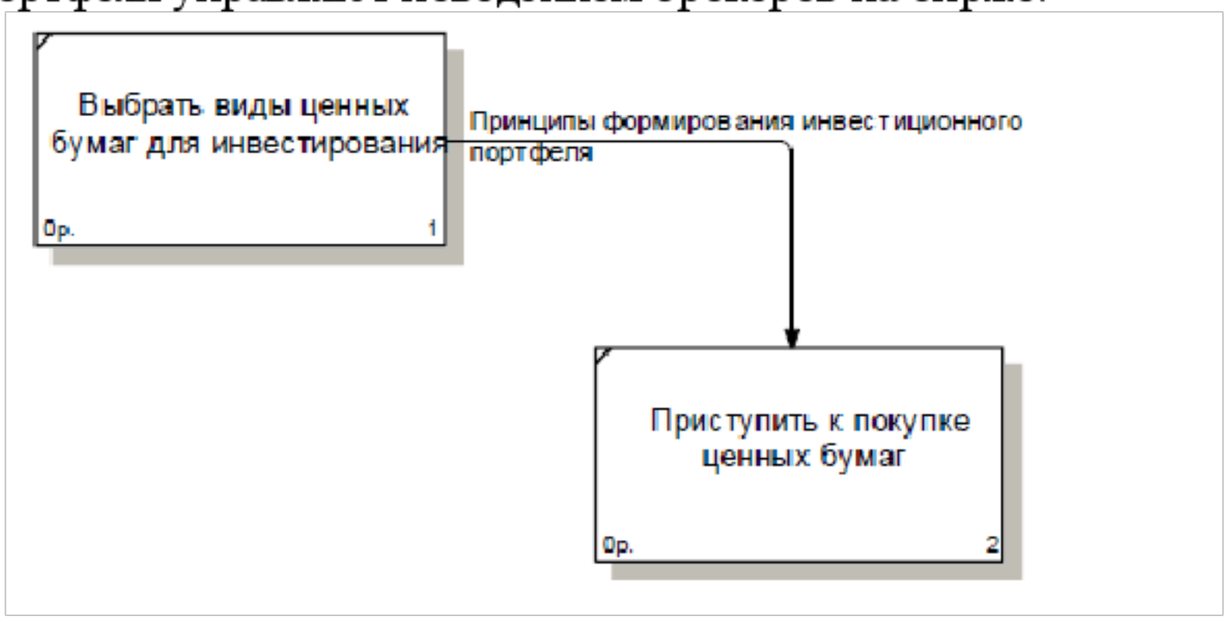


Рисунок 2.21 - Комбинированная стрелка «Выход – управление»

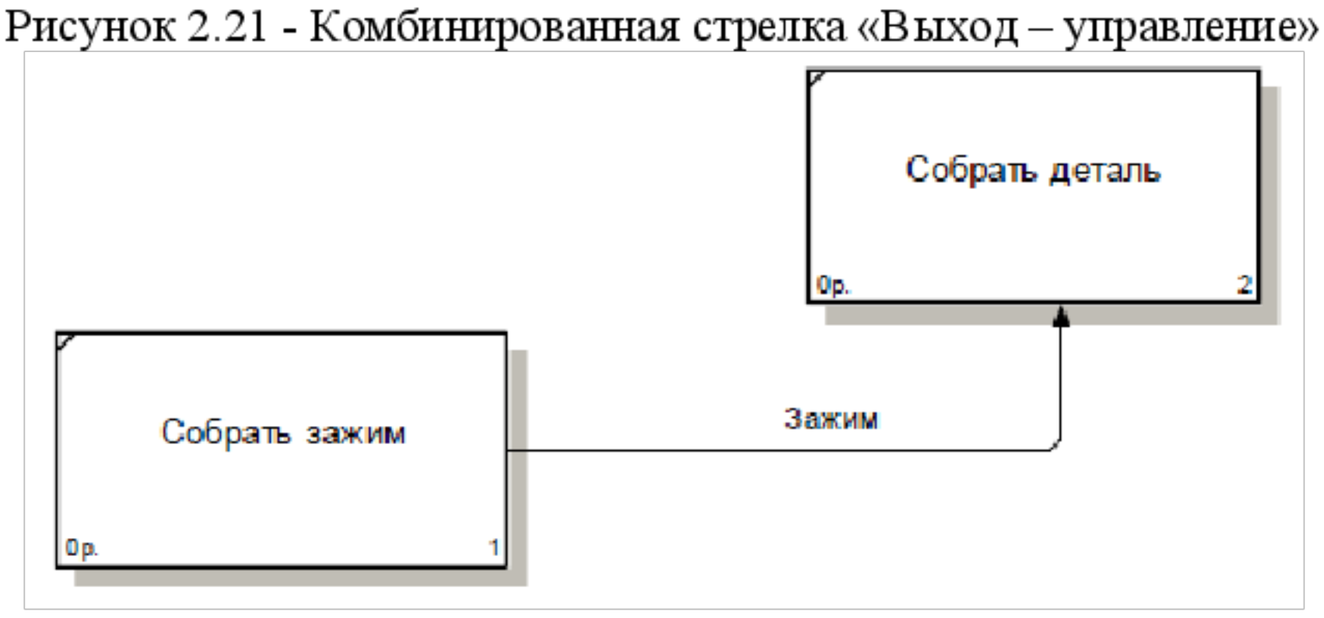


Рисунок 2.22 - Комбинированная стрелка «Выход – механизм исполнения»

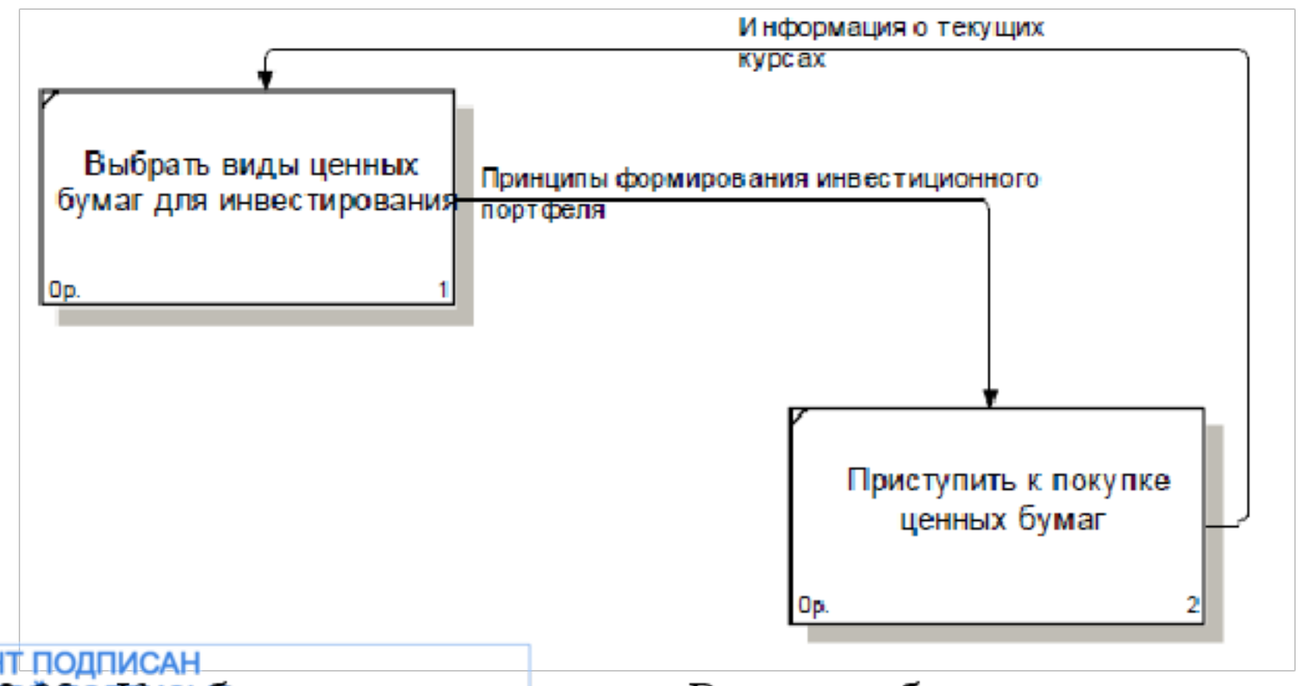


Рисунок 2.23 - Комбинированная стрелка «Выход – обратная связь на управление»

Стрелки «Выход – механизм исполнения» встречаются реже и отражают ситуацию, когда выход одного функционального блока применяется в качестве оборудования для работы другого

блока. На рисунке 2.22 зажим, устройство, используемое для закрепления детали во время ее сборки, должно быть собрано для того, чтобы выполнить сборку детали.

Обратные связи на вход и на управление применяются в случаях, когда зависимые блоки формируют обратные связи для управляющих ими блоков.

На рисунке 2.23 получаемая от брокеров информация о текущих биржевых курсах применяется для корректировки стратегии игры на бирже.

Стрелка «Выход – обратная связь на вход» обычно применяется для описания циклов повторной обработки чего-либо. Рисунок 2.24 может служить примером применения стрелки такого типа. Кроме того, связи «Выход – обратная связь на вход» могут применяться в случае, если бракованная продукция может заново использоваться в качестве сырья, как это происходит, например, при производстве оконного стекла, когда разбитое в процессе производства стекло перемалывается и переплавляется заново вместе с обыкновенным сырьем.

Разбиение и соединение стрелок. Выход функционального блока может использоваться в нескольких других блоках. Фактически чуть ли не главная ценность IDEF0 заключается в том, что эта методология помогает выявить взаимозависимости между блоками системы. Соответственно IDEF0 предусматривает как разбиение, так и соединение стрелок на диаграмме. Разбитые на несколько частей стрелки могут иметь наименования, отличающиеся от наименования исходной стрелки. Исходная и разбитые (или объединенные) стрелки в совокупности называются связанными. Такая техника обычно применяется для того, чтобы отразить использование в процессе только части сырья или информации, обозначаемых исходной стрелкой (рисунок 2.25). Аналогичный подход применяется и к объединяемым стрелкам.

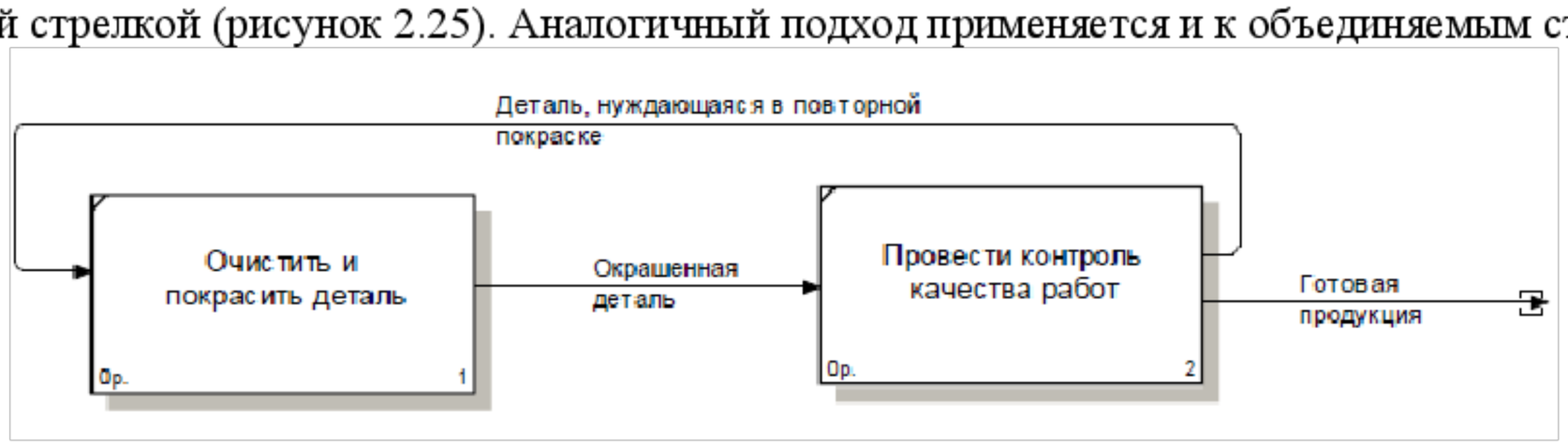


Рисунок 2.24 - Комбинированная стрелка «Выход – обратная связь на вход»

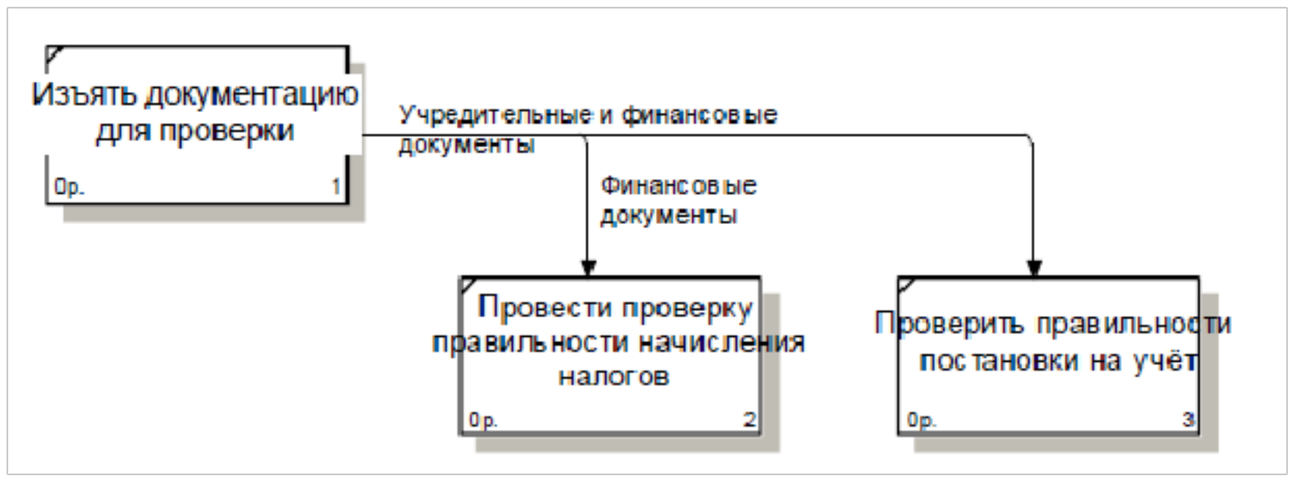


Рисунок 2.25 - Разбитая на две части и переименованная стрелка

Туннели. Понятие «связанные стрелки» используется для управления уровнем детализации диаграмм. Если одна из стрелок диаграммы отсутствует на родительской диаграмме (например, ввиду своей несущественности для родительского уровня) и не связана с другими стрелками той же диаграммы, точка входа этой стрелки на диаграмму или выхода с нее обозначается туннелем. На рисунок 2.26 , например, стрелка «Корпоративная информационная система» – важный механизм исполнения для данной диаграммы, но, возможно, она более нигде не используется в модели. Туннель в данном случае используется как альтернатива загромождению родительских диаграмм помещением на них несущественных для их уровня стрелок.

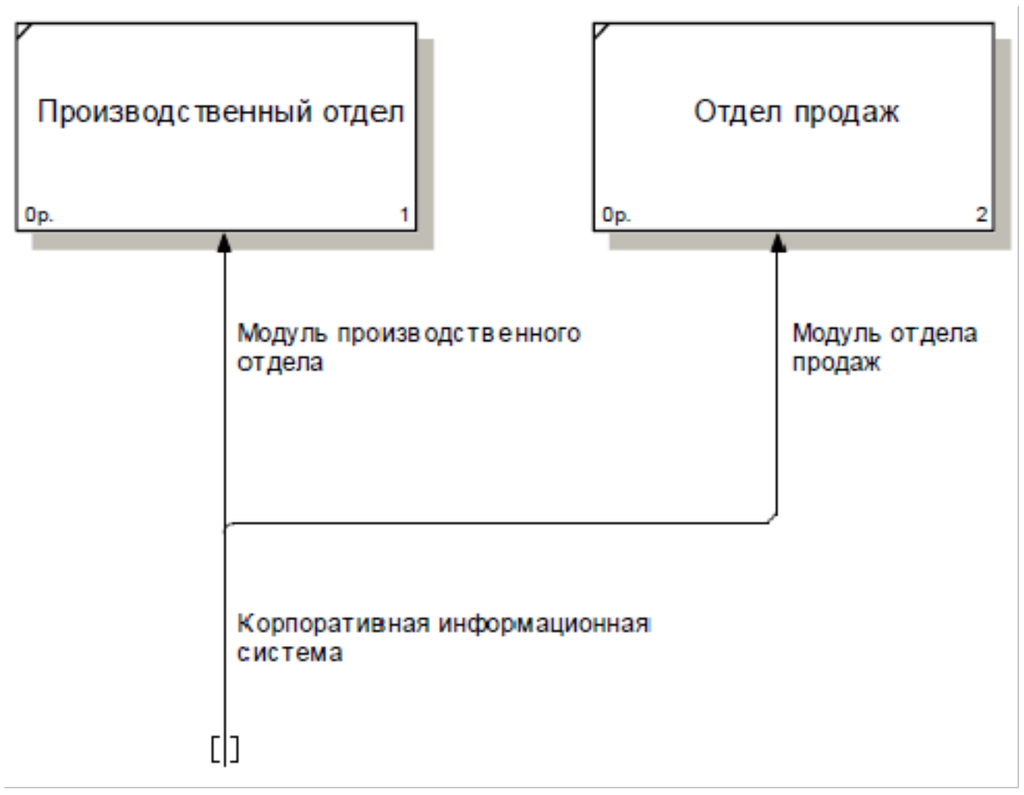


Рисунок 2.26 - Пример применения туннеля

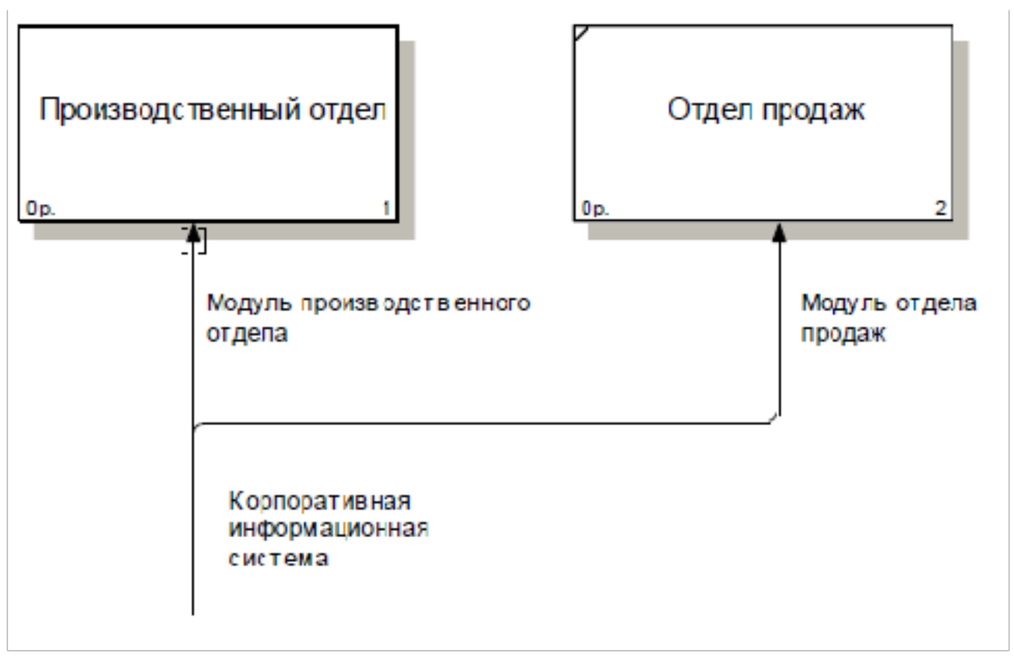


Рисунок 2.27- Еще один пример применения туннеля

Кроме того, туннели применяются для отражения ситуации, когда стрелка, присутствующая на родительской диаграмме, отсутствует в диаграмме декомпозиции соответствующего блока. На рисунке 2.27 туннель у стрелки «Модуль производственного отдела» обозначает, что на диаграмме декомпозиции производственного отдела отсутствует стрелка механизма управления с соответствующим наименованием.

2.4.2.2 Построение моделей IDEF0

Здесь мы рассмотрим методику построения моделей IDEF0 более подробно. Диаграммы. На рисунке 2.28 типовая диаграмма IDEF0 показана вместе с находящейся на ее полях служебной информацией. Служебная информация состоит из хорошо выделенных верхнего и нижнего колонтитулов (заголовка и «подвала»). Элементы заголовка используются для отслеживания процесса создания модели. Элементы «подвала» отображают наименование модели, к которой относится диаграмма, и показывают ее расположение относительно других диаграмм модели.

Рисунок 2.28 - IDEF0-диаграмма со служебной информацией на полях

Все элементы заголовка диаграммы перечислены в таблице 2.12.

Таблица 2.12 - Элементы заголовка диаграммы IDEF0

Поле	Назначение
USED AT	Используется для отражения внешних ссылок на данную диаграмму (заполняется на печатном документе вручную)
Author, date, project	Содержит Ф. И. О. автора диаграммы, дату создания, дату последнего внесения изменений, наименование проекта, в рамках которого она создавалась
Notes 1 ... 10	При ручном редактировании диаграмм пользователи могут зачеркивать цифру каждый раз, когда они вносят очередное исправление
Status	Статус отражает состояние разработки или утверждения данной диаграммы. Это поле используется для реализации формального процесса публикации с шагами пересмотра и утверждения
Working	Новая диаграмма, глобальные изменения или новый автор для существующей диаграммы
Draft	Диаграмма достигла некоторого приемлемого для читателей уровня и готова для представления на утверждение
Recommended	Диаграмма одобрена и утверждена, какие-либо изменения не предвидятся
Publication	Диаграмма готова для окончательной печати и публикации
Reader	Ф. И. О. читателя
Date	Дата знакомства читателя с диаграммой
Context	Набросок расположения функциональных блоков на родительской диаграмме, на котором подсвечен декомпозируемый данной диаграммой блок. Для диаграммы самого верхнего уровня (контекстной диаграммы) в поле помещается контекст TOP

Все элементы нижней строки диаграммы перечислены в таблице 2.13.

Таблица 2.13 - Элементы нижней строки диаграммы IDEF0

Поле	Назначение
Node	Номер диаграммы, совпадающий с номером родительского функционального блока
Title	Имя родительского функционального блока
Number	Уникальный идентификатор данной версии данной диаграммы. Таким образом, каждая новая версия данной диаграммы будет иметь новое значение в этом поле. Как правило, C-Number состоит из инициалов автора (которые предполагаются уникальными среди всех аналитиков проекта) и последовательного уникального идентификатора, например SDO005. При публикации эти номера могут быть заменены стандартными номерами страниц. Если диаграмма замещает другую диаграмму, номер заменяемой диаграммы может быть заключен в скобки – SDO005 (SDO004). Это обеспечивает хранение истории изменений всех диаграмм модели

Цикл «эксперт – аналитик». Подобно циклу «автор – редактор», применяющемуся в книгоиздательском деле, диаграммы IDEF0 пересматриваются и изменяются для

обеспечения точности отражения предметной области и улучшения своего качества.

Для каждого рецензента автором, как правило, подготавливается свой набор диаграмм. Предложения по изменениям и исправлениям возвращаются рецензентами автору для внесения их в модель. При возникновении разногласий между автором и рецензентом спорная диаграмма обычно рассылается всем рецензентам для достижения группового консенсуса.

Формально механизм рецензирования и модификации диаграмм поддерживается полями Status и нумерацией диаграмм, контроль истории изменений – полем Field (см. таблицу 2.12).

Построение моделей. Ни одна модель не должна строиться без ясного осознания объекта и целей моделирования. Выбранное определение цели моделирования должно отвечать на следующие вопросы:

Почему моделируется данный процесс?

Что выявит данная модель?

Как ознакомившиеся с этой моделью смогут ее применить?

Следующее предложение может служить примером формулирования цели моделирования. Выявить задачи каждого работника компании и понять в целом взаимосвязь между отдельно взятыми задачами для разработки руководства по обучению новых сотрудников.

Модели строятся для того, чтобы ответить на набор поставленных вопросов. Такие вопросы формулируются на ранних стадиях моделирования и впоследствии служат основой для четкого и краткого определения цели моделирования. Примерами таких вопросов могут быть:

Каковы задачи менеджера?

Каковы задачи клерка?

Кто контролирует работу?

Какая технология нужна для выполнения каждого шага? и т. п.

Точка зрения. С методической точки зрения при моделировании полезно использовать мнение экспертов, имеющих разные взгляды на предметную область, однако каждая отдельно взятая модель должна разрабатываться исходя из единственной заранее определенной точки зрения. Часто другие точки зрения вкратце документируются в прикрепленных диаграммах FEO (см. ниже) исключительно для наглядности изложения.

Точку зрения нужно подбирать достаточно аккуратно, основой для выбора должна служить поставленная цель моделирования. Наименованием точки зрения может быть наименование должности, подразделения или роли (например, руководитель отдела или менеджер по продажам). Как и в случае с определением цели моделирования, четкое определение точки зрения необходимо для обеспечения внутренней целостности модели и предотвращения постоянного изменения ее структуры. Может оказаться необходимым построение моделей с разных точек зрения для детального отражения всех особенностей, выделенных в системе функциональных блоков.

Границы моделирования. Одним из положительных результатов построения функциональных моделей оказывается прояснение границ моделирования системы, а в целом и ее основных компонентов. Хотя и предполагается, что в процессе работы над моделью будет происходить некоторое изменение границ моделирования, их вербальное (словесное) описание должно поддерживаться с самого начала для обеспечения координации работы участвующих в проекте аналитиков. Как и при определении цели моделирования, отсутствие границ затрудняет оценку степени завершенности модели, поскольку границы моделирования имеют тенденцию к расширению с ростом размеров модели.

Границы моделирования имеют два компонента: ширину охвата и глубину детализации. Ширина охвата обозначает внешние границы моделируемой системы. Глубина детализации определяет степень подробности, с которой нужно проводить декомпозицию функциональных блоков.

Чтобы облегчить правильное определение границ моделирования при разработке моделей IDEF0, существенные усилия затрачиваются на разработку и рецензирование контекстной диаграммы IDEF0 (диаграммы «самого верхнего» уровня). Иногда даже прибегают к построению дополнительной диаграммы для отображения уровня, более высокого, чем контекстный, для данной модели, что позволяет обозначить систему, внутри которой располагается объект для моделирования. Существенные затраты на разработку контекстной диаграммы вполне оправданы, поскольку она является своего рода «точкой отсчета» для остальных диаграмм модели и вносимые в нее изменения каскадом отражаются на всех лежащих

ниже уровнях.

Когда границы моделирования понятны, становятся ясными и причины, по которым некоторые объекты системы не вошли в модель.

Выбор наименования контекстного блока. Рекомендуемой последовательностью действий при построении модели с нуля являются: формулирование цели моделирования, выбор точки зрения, определение границ моделирования. Наименование контекстного блока – функционального блока самого высокого уровня – обобщает определение границ моделирования.

Правила подбора имени для контекстного блока в целом не отличаются от общих правил наименования функциональных блоков, поэтому для них обычно подбирают обобщающие названия, типа «Управление отделом по работе с клиентами», «Обработка заказов» и т. п.

Определение стрелок на контекстной диаграмме. Стрелки диаграмм IDEF0 обычно проще проектировать в следующем порядке: выход, вход, механизм исполнения, управление. Каждый функциональный блок обозначает отдельную функцию, и эта функция часто имеет ясно и кратко описываемые результаты работы. Наличие неясностей при анализе выходов того или иного функционального блока – возможный сигнал необходимости проведения ре- инжиниринга рассматриваемого бизнес-процесса.

Определение выходов. После идентификации возможных выходов полезно провести анализ модели на предмет покрытия ею всех возможных сценариев поведения процесса. Это означает, что если существует вероятность возникновения той или иной ситуации в ходе процесса, модель отражает возможность возникновения такой ситуации. Многие начинающие аналитики забывают отразить негативные результаты работы функциональных блоков. Например, блок «Провести экзамен по вождению» определенно произведет поток водителей, только что получивших права, но вполне правомерно ожидать и потока лиц, не сдавших экзамен. Негативные результаты часто используются в качестве обратных связей, анализ на их наличие должен проводиться для каждого блока. Важным также является необходимость включения в модель спорных стрелок, принятие решения о наличии которых в модели вполне можно переложить на плечи рецензирующих модель экспертов.

Определение входов. Входы можно рассматривать как особым образом преобразуемые функциональными блоками для производства выхода сырье или информацию. В производственных отраслях определить, как входное сырье преобразуется в готовую продукцию, обычно довольно просто. Однако при моделировании информационных потоков входной поток данных может представляться не потребляемым и не обрабатываемым вообще. Случаи, когда входящие и исходящие стрелки называются в точности одинаково, крайне редки и в основном указывают на бесполезность данного блока для системы в целом или на некорректный выбор имени для исходящей стрелки. Решением может служить применение более подробного описания для входящих и исходящих потоков данных. Например, вход может иметь название «Предварительный диагноз пациента», а выход – «Уточненный диагноз пациента».

Определение механизмов исполнения. После создания входов и выходов можно приступить к рассмотрению механизмов исполнения, или ресурсов, относящихся к функциональному блоку. В понятие механизма исполнения входят персонал, оборудование, информационные системы и т. п. Например, функциональный блок «Собрать деталь» может потребовать использования какого-либо оборудования, например гаечного ключа. При приеме экзаменов на водительские права механизмом исполнения является инспектор ГИБДД. Как правило, определить механизмы исполнения для функциональных блоков довольно просто.

Определение управления. Должно быть определено управление, контролирующее ход работы функционального блока. Все функциональные блоки в IDEF0 должны иметь хотя бы одно управление. В случаях, когда не ясно, относить ли стрелку к входу или к управлению, следует ее рисовать как управление. Важно помнить, что управление можно рассматривать как особую форму входа функционального блока.

Когда контекстная диаграмма представляется завершенной, попробуйте задать следующие вопросы:

Обобщает ли диаграмма моделируемый бизнес-процесс?

Согласуется ли диаграмма с границами моделирования, точкой зрения и целью моделирования?

Подходит ли выбранный уровень детализации стрелок для контекстного блока? (Обычно на контекстной диаграмме рекомендуется рисовать не более шести стрелок каждого типа.)

Нумерация блоков и диаграмм. Все функциональные блоки IDEF0 нумеруются. В номерах допускается использование префиксов произвольной длины, но в подавляющем большинстве моделей используется префикс А. Номер блока проставляется за префиксом. Контекстный блок всегда имеет номер А0.

Префикс повторяется для каждого блока модели. Номера используются для отражения уровня декомпозиции, на котором находится блок. Блок А0 декомпозируется в блоки А1, А2, А3 и т. д. А1 декомпозируется в А11, А12, А13 и т. д. А11 декомпозируется в А111, А112, А113 и т. д. Для каждого уровня декомпозиции в конец номера добавляется одна цифра.

Связь между диаграммой и ее родительским функциональным блоком. Функциональный блок декомпозируется, если необходимо детально описать его работу. При декомпозиции блока полезно рассмотреть его жизненный цикл, это поможет определить функциональные блоки получающейся детской диаграммы. Например, жизненный цикл 1 блока «Поджарить бифштекс» может выглядеть как следующая последовательность: подготовить продукты – отбить мясо – разогреть масло и т. д.

При моделировании IDEF0 важно иметь в виду, что граница детской диаграммы есть граница родительского функционального блока. Это означает, что вся работа выполняется блоками самого нижнего уровня. В отличие от иерархии, применяемой в структурном программировании, блоки верхнего уровня не являются субъектами управления для блоков нижнего уровня. Это означает, что в IDEF0 дети – это те же объекты, что и их родители, только показанные с большей детализацией. Действия генерального директора компании на диаграммах IDEF0 могут отражаться рядом с действиями простых рабочих.

На концах граничных стрелок (начинающихся или заканчивающихся за пределами диаграммы) детских диаграмм помещаются коды ICOM, чтобы показать, где находится соответствующая стрелка на родительской диаграмме (рисунок 2.29). Они нужны для проверки целостности модели и могут быть полезны, когда порядок расположения стрелок на детской диаграмме отличается от порядка их расположения на родительской диаграмме. Код ICOM состоит из латинской буквы I, C, O или M и числа, показывающего расположение стрелки на родительской диаграмме в порядке сверху вниз или слева направо.

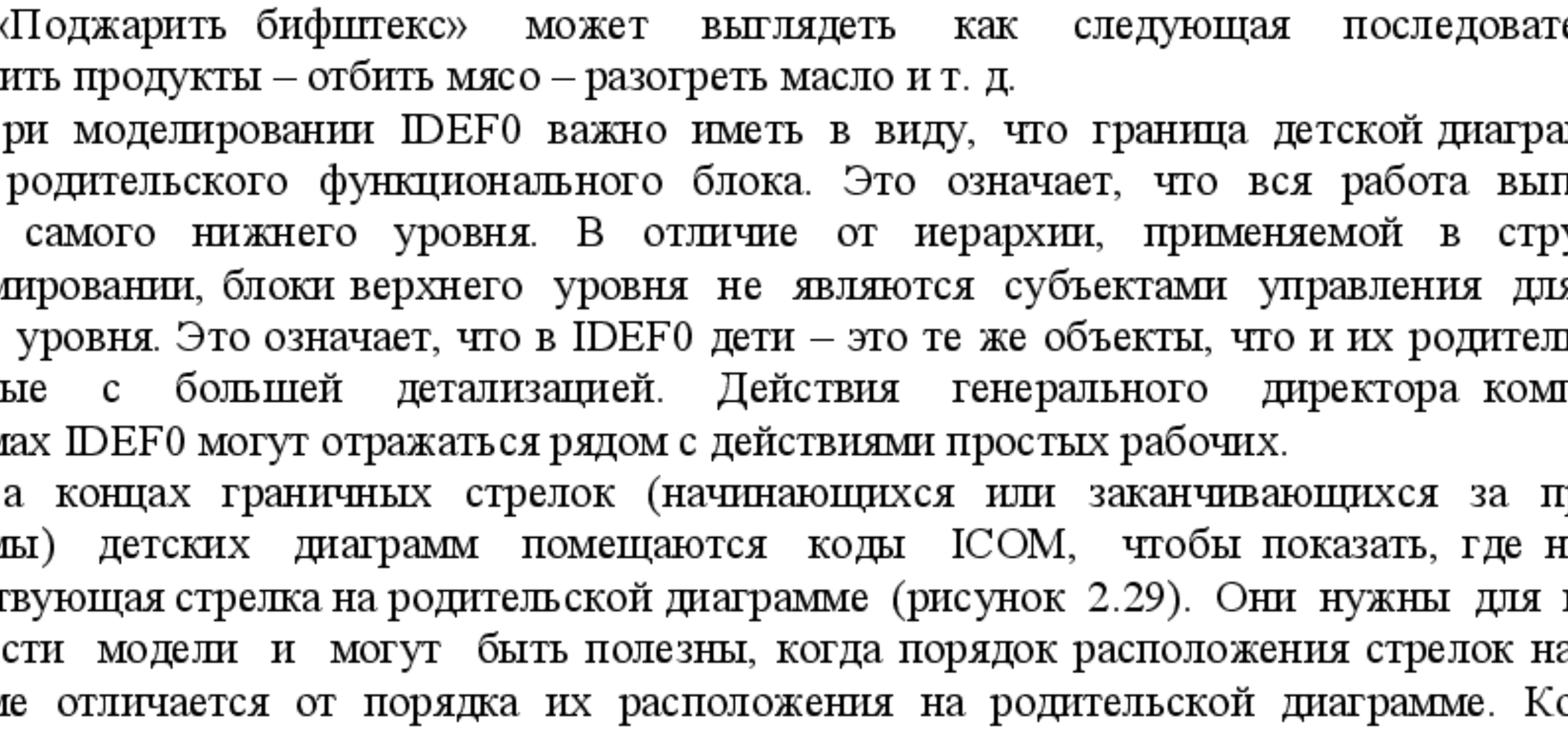


Рисунок 2.29 - ICOM-коды на граничных стрелках

Два подхода к началу моделирования («в ширину» и «в глубину»). Модели могут проектироваться с использованием подхода «в ширину», когда каждая диаграмма максимально детализируется перед своей декомпозицией, и с подходом «в глубину», когда сначала определяется иерархия блоков, а затем создаются соединяющие их стрелки. Естественно, возможно применение комбинации этих подходов, причем иерархия блоков может иногда немного измениться после того, как нарисованы стрелки. Это происходит из-за того, что создание стрелок может изменить понимание.

План конспекта:

1. Введение в системный анализ.
2. Анализ проблемы и моделирование предметной области с использованием системного подхода.
3. Методология ARIS. Стандарты IDEF0 – IDEF3.

Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-3	1-3	1-3	1-2

Тема самостоятельного изучения № 3
Анализ требований и их формализация.

Вид деятельности студентов: самостоятельное изучение литературы
Итоговый продукт самостоятельной работы: конспект
Средства и технологии оценки: собеседование
Краткие теоретические сведения

3 АНАЛИЗ ТРЕБОВАНИЙ И ИХ ФОРМАЛИЗАЦИЯ

3.1 Методы определения требований

3.1.1 Интервьюирование

Одним из наиболее важных и понятных методов получения требований является *интервью с клиентом;* это метод, который можно использовать практически в любой ситуации.

Одна из основных задач интервьюирования – сделать все возможное, чтобы предубеждения и предпочтения интервьюируемых не повлияли на свободный обмен информацией. Это сложная проблема. Социология учит нас, что невозможно воспринимать окружающий мир, не фильтруя его в соответствии со своим происхождением и накопленным опытом.

3.1.1.1 Этапы проведения интервью

Процесс проведения интервью состоит из следующих этапов:

Разработка вопросов (образец вопросов интервью представлен в таблице 3.1).

Выбор опрашиваемых пользователей. Существует несколько групп пользователей: персонал начального уровня, организаторы проекта среднего уровня, менеджеры и другие заказчики особого рода – генеральные директоры и вице-президенты, научные сотрудники, обычные пользователи системы.

Планирование контактов.

Проведение интервью. Интервью можно провести по телефону, персонально или с помощью Интернета (видеоконференция, чат, электронная почта), однако лучшим способом осуществления интервью является непосредственное общение, лицом к лицу.

Завершение встречи и определение последующих действий.

Таблица 3.1 - Обобщенное практически контекстно-свободное интервью

Часть I. Определение профиля заказчика или пользователя
Имя
Компания
Отрасль
Должность
(Вышеприведенная информация, как правило, может быть внесена заранее.)
Каковы ваши основные обязанности?
Что вы в основном производите?
Для кого?
Как измеряется успех вашей деятельности?
Какие проблемы влияют на успешность вашей деятельности?
Какие тенденции, если такие существуют, делают вашу работу проще или сложнее?
Часть II. Оценка проблемы
Для каких проблем (прикладного типа) вы ощущаете нехватку хороших решений?
Назовите их. (Замечание. Не забывайте спрашивать: «А еще?».)
Для каждой проблемы выясняйте следующее.
Почему существует эта проблема?
Как она решается в настоящее время?
Как заказчик (пользователь) хотел бы ее решать?
Часть III. Понимание пользовательской среды
Кто такие пользователи?
Какое у них образование?
Каковы их навыки в компьютерной области?
Имеют ли пользователи опыт работы с данным типом приложений?
Какая платформа используется?
Каковы ваши планы относительно будущих платформ?
Используются ли дополнительные приложения, которые имеют отношение к данному приложению? Если да, то пусть о них немного расскажут.
Каковы ожидания заказчика относительно практичности продукта?
Сколько времени необходимо для обучения?

В каком виде должна быть представлена справочная информация для пользователя (в интерактивном или в виде печатной копии)?

Часть IV. Резюме (перечисляются основные пункты, чтобы проверить, все ли правильно вы поняли)

Итак, вы сказали мне (перечислите описанные заказчиком проблемы своими словами)

Адекватно ли этот список представляет проблемы, которые имеются при существующем решении?

Какие еще проблемы (если такие существуют) вы испытываете?

Часть V. Предположения, аналитика относительно проблемы заказчика (проверенные или непроверенные предположения)

(те проблемы, которые не были упомянуты)

Какие проблемы, если они есть, связаны с (перечислите все потребности или дополнительные проблемы, которые, по-вашему, может испытывать заказчик или пользователь)

Для каждой из указанных проблем выясните следующее.

Является ли она реальной?

Каковы ее причины?

Как она решается в настоящее время?

Как бы заказчик (пользователь) хотел ее решать?

Насколько важно для заказчика (пользователя) решение этой проблемы в сравнении с другими, упомянутыми им?

Часть VI. Оценка предлагаемого вами решения (если это уместно)

(Охарактеризуйте основные возможности предлагаемого вами решения.
А потом задайте пользователю следующие вопросы.)

Что, если вы сможете?

Как вы расцениваете важность этого?

Часть VII. Оценка возможности

Кто в организации нуждается в данном приложении?

Сколько пользователей указанных типов будет использовать его?

Насколько значимо для вас успешное решение?

Часть VIII. Оценка необходимого уровня надежности и производительности, а также потребности в сопровождении

Каковы ваши ожидания относительно надежности?

Какой, по-вашему, должна быть производительность?

Будете ли вы заниматься поддержкой продукта или этим будут заниматься другие?

Испытываете ли вы потребности в поддержке?

Что вы думаете о доступе для сопровождения и обслуживания?

Каковы требования относительно безопасности?

Какие требования относительно установки и конфигурации?

Существуют ли специальные требования по лицензированию?

Как будет распределено программное обеспечение?

Есть ли требования на маркировку и упаковку?

Часть IX. Другие требования

Существуют ли законодательные требования, требования информационной среды, инструкции или другие стандарты, которых необходимо придерживаться? Нет ли других требований, о которых нам следовало бы знать?

Часть X. Окончание

Существуют ли другие вопросы, которые мне следовало бы вам задать?

Если мне еще понадобится задать вам несколько вопросов, могу ли я вам позвонить?

Будете ли вы принимать участие в обсуждении требований?

Часть XI. Заключение аналитика

После интервью, пока его данные еще свежи в вашей памяти, зафиксируйте три потребности или проблемы с наивысшими приоритетами, выявленные вами в беседе с данным заказчиком (пользователем)

3.1.2 «Мозговой штурм» и отбор идей

«Мозговой штурм» – это метод проведения собрания, при котором группа людей пытается найти решение специфической проблемы посредством накопления всех спонтанно возникающих идей.

Данный процесс имеет ряд очевидных преимуществ. Поддерживает участие всех присутствующих. Позволяет участникам развивать идеи друг друга.

Ведущий или секретарь ведет запись всего хода обсуждения.

Его можно применять при различных обстоятельствах.

Как правило, в результате получаем множество возможных решений для любой поставленной проблемы.

Метод способствует свободному мышлению, не ограниченному обычными рамками.

«Мозговой штурм» состоит из двух фаз: генерация идей и их отбор.

Основная цель на этапе генерации состоит в том, чтобы описать как можно больше идей, не обязательно глубоких. На этапе отбора главной задачей является анализ всех возникших идей. Отбор идей включает в себя отсечение, организацию, упорядочение, развитие, группировку, уточнение и т. п.

3.1.2.1 Генерация идей

«Мозговой штурм» можно производить различными способами. Ниже описывается один из простых процессов. Все основные участники собираются в одной комнате, и им раздаются материалы для заметок. Это может быть просто стопка бумаги и черный толстый маркер. Листы бумаги должны быть не менее 7×12 см и не более 12×17 см. Каждому участнику нужно выдать не менее 25 листов на каждый сеанс «мозгового штурма».

Каждый участник сеанса «мозгового штурма» исполняет одну из трех ролей: лидер, секретарь или член команды. Лидер отвечает за направление процесса в правильное русло, за порядок и помогает секретарю делать запи- си. Секретарь записывает все идеи таким образом, чтобы каждый человек, находящийся в комнате, мог видеть эти записи. Участники генерируют идеи.

Ведущий объясняет правила проведения «мозгового штурма» (рисунок 3.1) и определяется цель заседания.

Правила проведения «мозгового штурма»
1. Не допускается критика или дебаты.
2. Дайте свободу фантазии.
3. Генерируйте как можно больше идей.
4. Переделывайте и комбинируйте идеи.

Рисунок 3.1- Правила проведения «мозгового штурма»

По мере создания идей секретарь просто собирает их и прикрепляет к стене комнаты заседаний. Большинство заседаний по генерации идей длится около часа (иногда 2–3 часа).

3.1.2.2 Отбор идей

После завершения фазы генерации идей начинается процесс отбора, который состоит из нескольких этапов.

1. Отсечение. Заключается в отсечении тех идей, которые не достойны внимания. Если обнаруживается две одинаковые идеи, эти идеи объединяются.

2. Группировка идей. Группы получают названия в зависимости от того, по какому принципу осуществляется группировка. Например, группы могут иметь следующие названия: новые функции, вопросы производительности, предложения по усовершенствованию существующих функций, интерфейс пользователя и вопросы простоты обращения и т. д. Для любой из этих групп можно возобновить генерацию идей, если окажется, что процесс группировки стимулировал возникновение новых идей или некоторая область важных функциональных возможностей осталась неохваченной.

3. Определение функций. Ведущий перечисляет все оставшиеся идеи и просит авторов дать их описание, состоящее из одного предложения (таблица 3.2).

4. Расстановка приоритетов.

Таблица 3.2 - Пример определения функций

Область применения приложения	Штурмуемая функция	Описание функции
Автоматизация домашнего освещения	Автоматическое задание освещения	Домовладелец может предварительно задавать основанные на времени последовательности возникновения определенных осветительных событий в зависимости от времени дня
Система ввода заказов на покупку	Быстрота	Достаточно быстрое время ответа, чтобы не мешать проведению обычных операций
Система обнаружения неполадок	Автоматическое уведомление	Все зарегистрированные стороны будут уведомлены посредством электронной почты, когда что-нибудь изменится

3.1.3 Совместная разработка приложений (JAD – Joint Application Design)

Метод совместной разработки приложений – это зарегистрированный товарный знак, относящийся к компании IBM. Он представляет собой групповой подход к определению требований, при реализации которых особое внимание уделяется усовершенствованию группового процесса и правильному подбору людей, вовлеченных в работу над проектом.

Сеансы JAD аналогичны сеансам «мозгового штурма», однако не во всем. Сеансы «мозгового штурма» длятся около двух часов, а сеансы JAD – до трех дней. На сеансах «мозгового штурма» происходит быстрое генерирование идей, а на сеансах JAD – высокоуровневые специфические программные модели функций, данных и линий поведения.

Сеанс JAD имеет определенную структуру, на нем придерживаются определенной дисциплины, и он проходит под руководством арбитра. В его основе лежит обмен информацией с использованием документации, фиксированных требований и правил работы. С момента появления методики JAD на сеансах используются CASE-инструменты и другие программные средства, предназначенные для построения диаграмм потока данных (Data flow dia- gram, DFD), диаграмм взаимосвязей между сущностями (Entity relationship diagrams, ERD), диаграмм смены состояний и других объектно-ориентированных диаграмм.

3.1.3.1 Роли в сеансах JAD

Разработчики. В задачу разработчика входит оказание помощи организаторам в формулировании их потребностей, которые обычно являются решением существующих проблем. Таким образом, определение требований к ПО происходит совместно с организаторами проекта.

Участники. Для успешного проведения сеанса ключевым требованием является высокий уровень квалификации приглашенных. Правильный подбор людей позволит быстро принимать решения и разрабатывать правильные модели, даже если они не будут завершены.

Арбитр/консультант. Арбитр должен сводить к минимуму проявление непродуктивных человеческих эмоций, наподобие агрессивности или самозащиты. Арбитр не является хозяином ни процесса, ни программного продукта. Он присутствует только для того, чтобы помогать организаторам проекта, разрабатывать программный продукт.

Секретарь. Секретарь сеанса JAD документирует идеи и помогает следить за временем.

3.1.3.2 Сеансы JAD

Согласно Вуду (Wood), сеанс JAD – это своеобразная мастерская, работающая в максимально напряженном режиме, где решения принимаются совместно всеми участниками. При этом участники являются крупными специалистами в рассматриваемом вопросе.

Процесс исследования проекта разбивается на следующие этапы: поиск фактов и сбор информации, подготовка сеанса JAD, проведение самого сеанса JAD и проверка собранной

информации.

3.1.3.3 Результаты проведения сеанса JAD

Результатами проведения сеанса могут быть:
- диаграмма контекста данных;
- диаграмма потока данных первого уровня;
- глобальная модель данных – диаграмма взаимосвязей между сущностями;
- перечень первичных объектов;
- объектная модель высокого уровня;
- обязанности кандидатов и сотрудников для каждого объекта;
- перечень первичных процессов/сценарии выбора;
- другие диаграммы потока данных, диаграммы состояния, деревья альтернатив, таблицы решений;
- требования, предъявляемые к данным для каждого процесса;
- перечень допущений;
- документация по анализу или назначению открытых вопросов.

Результаты сеанса JAD используются в процессе определения требований для организации следующего этапа – создания спецификации SRS.

3.1.3.4 Недостатки метода JAD

Например, участники передают свои идеи арбитру и/или секретарю. Во избежание неверной интерпретации собранных данных, необходимо принять некоторые меры предосторожности. Использование во время сеанса автоматизированных инструментальных средств и проверка результатов всеми участниками уменьшит риск.

На сеансах JAD рассматриваются преимущественно информационные системы, в которых особое внимание уделяется элементам данных и проекту интерфейса. Есть мало информации об использовании метода JAD для определения требований, предъявляемых к системам реального времени.

На проведение трехдневного сеанса JAD с представителями всех групп организаторов проекта, каждая из которых состоит из квалифицированных специалистов, уходит немало средств. Три дня – это средняя продолжительность. Для анализа сложных вложенных систем реального времени и систем, от которых зависит человеческая жизнь, часто требуется больше времени. Если сеанс длится «до тех пор, пока не устанем», то усталость может наступить уже в тот момент, когда будут определены только сценарии выбора.

3.1.4 Раскадровка

Целью раскадровки является получение ранней реакции пользователей на предложенные концепции приложения. С ее помощью можно на самых ранних этапах жизненного цикла наблюдать реакцию пользователей, до того как концепции будут превращены в код, а во многих случаях даже до разработки подробной спецификации.

Раскадровка имеет следующие преимущества:
- предельно недорога;
- дружественна пользователю, неформальна и интерактивна;
- обеспечивает ранний анализ пользовательских интерфейсов системы;
- легко создаваема и модифицируема.

Раскадровки можно использовать для ускорения концептуальной разработки различных граней приложения. Их можно применять для понимания визуализации данных, определения и понимания бизнес-правил, которые будут реализованы в новом бизнес-приложении, для определения алгоритмов и других математических конструкций, которые будут выполняться внутри встроенных систем, или для демонстрации отчетов и других результатов на ранних этапах. Раскадровки можно (и нужно!) использовать практически для всех приложений, в которых раннее получение реакции пользователей является ключевым фактором успеха.

3.1.4.1 Типы раскадровок

Раскадровки делятся на три типа в зависимости от режима взаимодействия с пользователем: пассивные, активные и интерактивные.

Пассивные представляют собой историю, рассказываемую пользователю. Они могут состоять из схем, картинок, моментальных копий экрана, презентаций PowerPoint или образцов выходной информации системы. В пассивной раскадровке аналитик играет роль системы и просто проводит пользователя по раскадровке, объясняя следующее: «Когда вы делаете это, происходит вот это».

Активные раскадровки обеспечивают автоматизированное описание *п*оведения системы при типовом использовании или в операционном сценарии. Они создаются с помощью анимации или автоматизации, возможно, посредством автоматического последовательного показа слайдов, анимационных средств или даже фильма.

Интерактивные дают пользователю опыт обращения с системой почти такой же реальный, как на практике. Для своего выполнения они требуют участия пользователя. Интерактивные раскадровки могут быть имитационными, в виде макетов или могут даже представлять собой отбрасываемый впоследствии код. Сложная интерактивная раскадровка, основанная на отбрасываемом коде, может быть весьма похожа на отбрасываемый прототип.

Как видно из рисунка 3.2, эти три типа раскадровки предлагают широкий спектр возможностей – от образцов выходной информации до «живых» демонстрационных версий. Различие между сложными раскадровками и ранними прототипами продукта весьма условно.

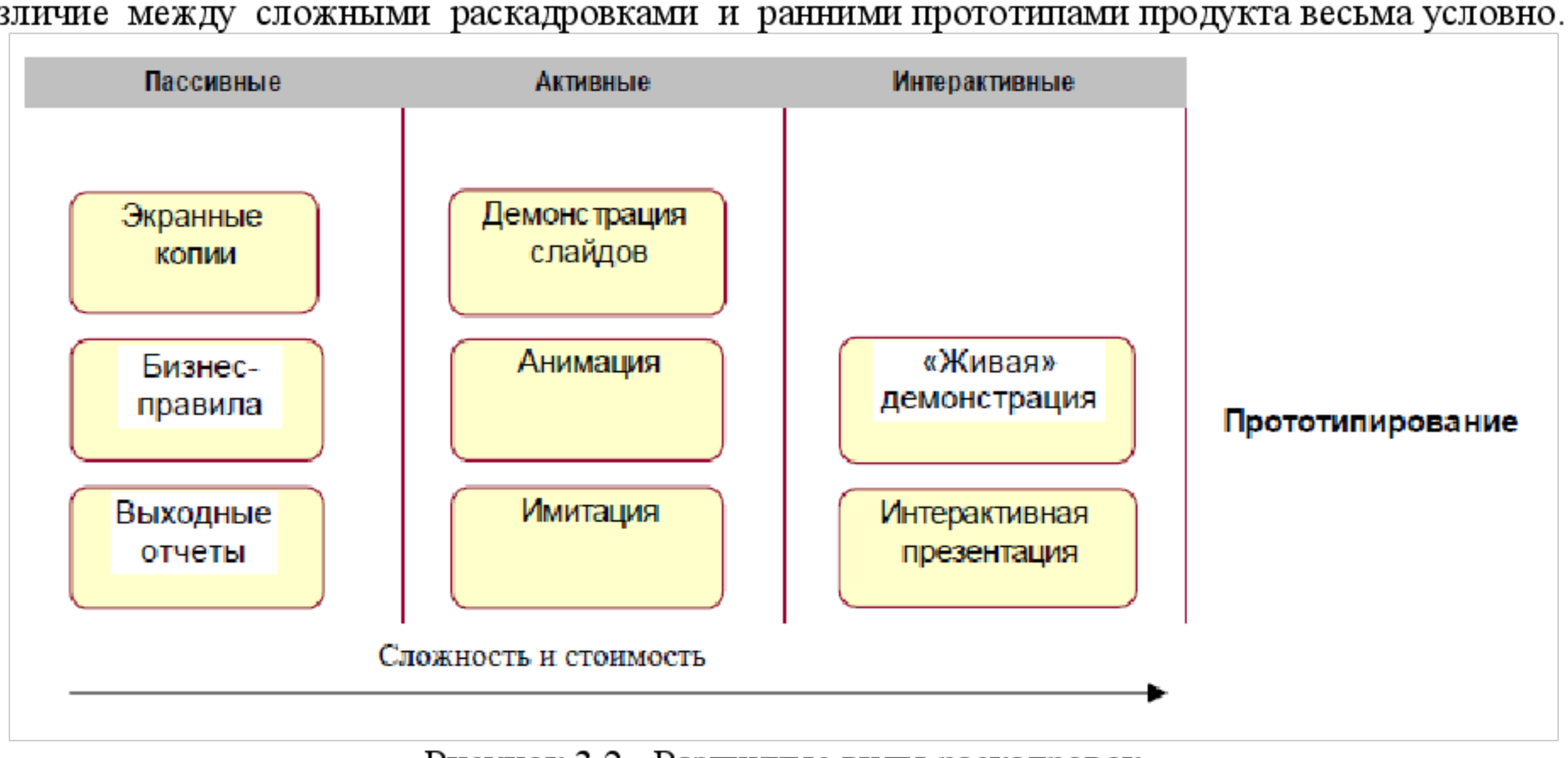


Рисунок 3.2 - Различные виды раскадровок

Выбор типа раскадровки зависит от сложности системы и того, насколько велик риск, что команда неправильно понимает ее назначение. Для беспрецедентной новой системы, имеющей расплывчатое определение, может потребоваться несколько раскадровок, от пассивной до интерактивной, по мере того как команда совершенствует свое понимание системы.

3.1.5 Обыгрывание ролей

Метод обыгрывания ролей позволяет команде разработчиков прочувствовать мир пользователя, побывав в его роли. Концепция, лежащая в основе данного метода, достаточно проста: хотя, наблюдая и задавая вопросы, мы повышаем уровень своего понимания, наивно полагать, что посредством одного наблюдения разработчик/аналитик может получить истинно глубокое понимание решаемой проблемы или требований к системе, которая призвана данную проблему решить.

3.1.5.1 Суть метода обыгрывания ролей

Аналитик или любой член команды занимает место пользователя и выполняет его

обычные действия. Рассмотрим в качестве примера проблему ввода заказов на покупку.

Разработчик/аналитик может «прочувствовать» проблемы и неточности, присущие существующей системе ввода заказов на покупку, просто заняв место оператора и попытавшись ввести несколько заказов. Полученный в течение часа опыт навсегда изменит понимание командой сути проблемы.

Существует две разновидности метода обыгрывания ролей: сценарный просмотр и CRC-карточки.

3.1.5.2 Сценарный просмотр

Сценарный просмотр – это исполнение роли на бумаге.

При сценарном просмотре каждый участник следует сценарию, который задает конкретную роль в «пьесе». Просмотр будет демонстрировать любые неточности в понимании ролей, недостаток информации, доступной актеру или подсистеме, или недостаток конкретного описания поведения, необходимого актерам для успешного выполнения их роли.

Одним из преимуществ сценарного просмотра является то, что сценарий можно модифицировать и проигрывать снова столько раз, сколько необходимо, пока актеры не сочтут его правильным. Сценарий можно также повторно использовать для обучения новых членов команды. Его можно модифицировать и проигрывать вновь, когда необходимо изменить поведение системы. В определенном смысле данный сценарий становится «живой» раскадровкой для проекта.

3.1.6 CRC-карточки (Class – Responsibility – Collaboration, класс – обязанность – взаимодействие)

Этот метод обыгрывания ролей часто применяется в объектно- ориентированном анализе. В данном случае каждому участнику выдается набор индексных карточек, описывающих класс (объект); обязанности (поведение); а также взаимодействия (с какими из моделируемых сущностей взаимодействует объект). Эти взаимодействия могут просто представлять сущности проблемной области (например, пользователи, нажатые кнопки, лампы и подъемники) или объекты, существующие в области решения (подсветка выключателя в холле, окно многодокументного интерфейса или кабина лифта).

Когда актер-инициатор инициирует определенное поведение, все участники следуют поведению, заданному на их карточках. Если процесс прерывается из-за недостатка информации или если одной сущности необходимо переговорить с другой, а взаимодействие не определено, то карточки модифицируются, и роли разыгрываются снова.

Ниже предлагается образец проигрывания одного из возможных вариантов использования.

1. **Управление включением.** Мой домовладелец только что нажал кнопку, управляющую набором ламп. Он все еще удерживает кнопку в нажатом состоянии. Я послал Центральному блоку управления сообщение, как только кнопка была нажата, и собираюсь посылать ему сообщения каждую секунду, пока кнопка нажата.

2. **Центральный блок управления.** Когда я получил первое сообщение, то изменил состояние выхода с «Выкл.» на «Вкл.». Когда я получил второе сообщение, стало очевидно, что домовладелец хочет изменить яркость освещения, поэтому при получении каждого сообщения я собираюсь изменять яркость на 10 %.

3. **Лампа.** Я аппаратно запрограммирован на изменяемый накал. Я изменяю яркость света при нашем разговоре.

3.1.7 Быстрое прототипирование

Быстрое прототипирование – это частичная реализация системы программного обеспечения, созданная с целью помочь разработчикам, пользователям и клиентам лучше понять требования к системе.

Чем детальнее прототип, тем легче понять требования заказчика. С другой стороны, прототипы сами по себе являются программами, поэтому, чем детальнее прототип, тем он дороже. Первый взгляд на проблему решения, строить прототип или нет, представлен на рисунке 3.3. Таблица показывает, например, что относительно недорогой прототип с большой

ценностью должен быть создан. Под большой ценностью имеется в виду, что построение прототипа поможет заказчику лучше понять, продукт какого типа будет выпущен, помогает программистам лучше понять, какой продукт они должны выпустить.

	Предполагаемая ценность прототипа	
	Низкая	Высокая
Низкая стоимость прототипа	Возможно	Да
Высокая стоимость прототипа	Нет	Возможно

Рисунок 3.3 - Выгода от прототипа: грубая оценка

Для случаев, когда однозначно нельзя определить, разрабатывать прототип или нет, нужно оценить затраты на разработку прототипа и возможную прибыль от его реализации. На основе этой оценки определяются оптимальные расходы на прототип и принимается решение о его разработке и размере финансирования в случае положительного решения (рисунок 3.4). По мере того как возрастают расходы на прототип, возрастает и его пригодность, но также возрастают и расходы из выделенного бюджета. В результате, вероятно, существует момент, в который затраты оптимальны (точка максимума на кри вой), и некоторая точка, за которой деньги уже потрачены зря (где кривая пересекает горизонтальную ось).

Существует много способов разбиения прототипов на категории. Выделяются следующие прототипы: отбрасываемые, эволюционирующие и операционные; вертикальные и горизонтальные; пользовательские интерфейсы и алгоритмические и т. д. Каким должен быть прототип в каждом конкретном случае, зависит от того, какую проблему вы пытаетесь решить путем построения прототипа.

Как пример представьте себе приложение для электронной коммерции, в котором компания-производитель одежды желает продавать товары через Интернет, хранить информацию о клиентах и предоставлять клиентам возможность получать свое фото в одежде из каталога. Финансовая оценка для разных уровней прототипирования для программы, продающей одежду, приведена в таблице 3.3. Для каждой из четырех характеристик, рассмотренных в прототипе, сделано несколько оценок: стоимость работы; процент работы, который будет повторно использоваться в самой программе (т. е. не будет отброшен); и полная прибыль от прототипа. Под полной прибылью здесь понимается оценка того, что будет получено, если характеристика будет включена в прототип, но код не будет использован в программе. Например, мы подсчитали, что если прототип «примерка одежды» будет построен, это сэкономит минимум 20 000 при разработке. Оценка базируется на нижеследующих факторах.

Предотвращение пустой траты времени на предложенные требования, которые, как видно из прототипа, не нужны (т. е. минимум три ненужных требования из 100; на этап требований выделено 300 000, сэкономлено 9000).

Разработка программного проекта «примерка одежды», что уменьшает риски разработки (т. е. оценка того, что это сэкономит минимум одну человеко-неделю времени проектирования = 2000).

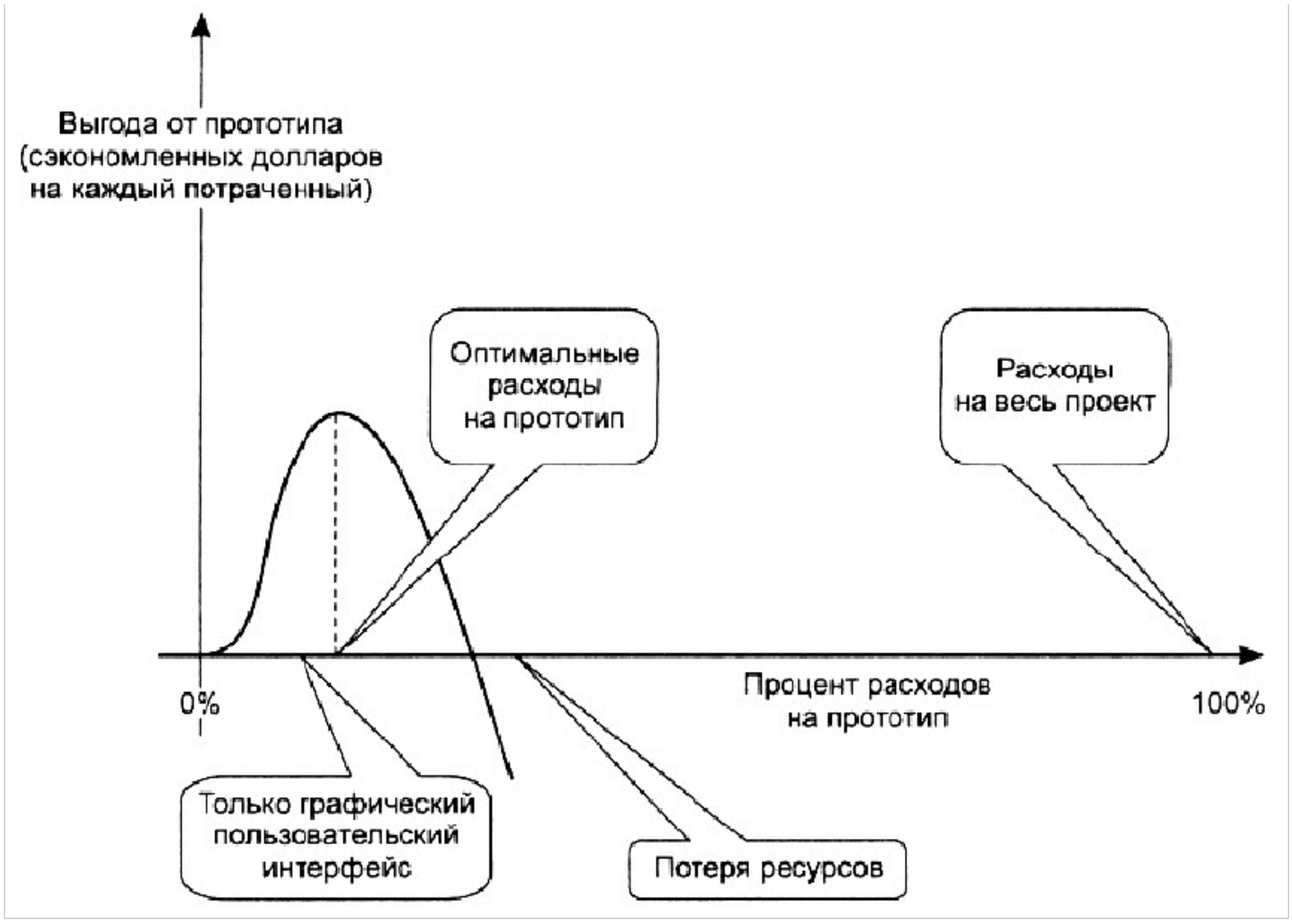


Рисунок 3.4 - Выгода от прототипа

Переработка, которая может возникнуть из-за изменения требований клиентом после того, как он увидит окончательный продукт (т. е. переработка минимум трех требований по 3000 каждое = 9000).

Существует минимальная экономия, эквивалентная 9000 + 2000 + 9000 = 20 000.

Оценка повторного использования кода может быть выполнена путем определения классов прототипа и решения того, какие из них будут использованы в самой программе.

Такая оценка состоит из исследования стоимости небольших частей, что все же является сложной задачей. Определение минимума и максимума такой оценки может несколько упростить этот трудный процесс.

Как только оценки сделаны, выполняется несложная часть вычисления лучшего и худшего сценария (таблица 3.3). Минимальное значение выгоды получается из самой пессимистичной комбинации – высоких затрат, низкой общей прибыли и низкого процента повторного использования. Максимальная выгода рассчитывается аналогично.

Усреднение – это один из способов работы с разницей между худшими и лучшими случаями. В результате получаем положительную выгоду для всех предложенных частей прототипа за исключением части «примерка одежды», которая оценена в –4000: общий убыток 4000. Это приведет в дальнейшем к относительно низкой прибыли, высокой стоимости разработки и низкому вторичному использованию.

Таблица 3.3 - Оценка программы по продаже одежды

Часть прототипа	Оценка стоимости	Общая прибыль без повторного использования кода		Процентная часть кода прототипа, повторно использованная в приложении	Чистая выгода		
		min	max		min	max	В среднем
	B	D	E	C	D – (1 – – C)B	E – (1 – – C)B	
1. Экранные снимки пользовательского интерфейса	10 000	10 000	80 000	50 %	5 000	75 000	40 000
2. Безопасность транзакций	50 000	10 000	300 000	80 %	0	290 000	145 000
3. Завершение транзакций	80 000	10 000	400 000	50 %	–30 000	200 000	85 000
4. Примерка одежды клиентом	120 000	20 000	140 000	30 %	–64 000	56 000	–4 000

3.2 Формализация требований

Иногда присущая естественному языку неоднозначность просто неприемлема, особенно когда требования касаются жизненно важных вопросов или когда неправильное поведение системы может привести к чрезвычайным экономическим или юридическим последствиям. Если определение требования сложно сформулировать на естественном языке и невозможно предотвратить неправильное понимание спецификации, следует попытаться написать эту часть требований с помощью теоретически обоснованных формальных методов.

3.2.1 Метод вариантов использования и его применение

Метод вариантов использования (прецедентов) является частью методологии объектно-ориентированного проектирования. Это метод анализа и проектирования сложных систем, представляющий собой способ описания поведения системы с точки зрения того, как различные пользователи взаимодействуют с ней для достижения своих целей. Такой ориентированный на пользователя подход предоставляет возможность исследовать различные варианты поведения системы при раннем привлечении пользователя.

Варианты использования можно успешно применять на протяжении всего жизненного цикла программного обеспечения: при анализе, проектировании и в процессе тестирования. В этом случае, процесс разработки программного обеспечения называют «основанный на вариантах использования».

Вариант использования – это функциональный связный блок, выраженный в виде транзакции между актантом и системой. Вариант использования описывает последовательность действий, выполняемых системой с целью предоставить полезный результат конкретному актанту.

3.2.1.1 Построение модели вариантов использования

Модель вариантов использования системы состоит из всех актантов системы и различных вариантов использования, посредством которых актанты взаимодействуют с системой, тем самым, описывая многообразие ее функционального поведения. Она также показывает связи между вариантами использования, что углубляет наше понимание системы.

Первый шаг моделирования вариантов использования состоит в создании системной диаграммы, описывающей границы системы и определяющей ее актанты. Это позволяет параллельно осуществить этапы 3 и 4, в которых требуется выявить заинтересованных лиц системы и определить ее границы.

Второй шаг описывает системное поведение, т. е. то, как пользователи взаимодействуют с системой, осуществляя некоторые последовательности действий, для достижения определенных

целей (рисунок 3.5).

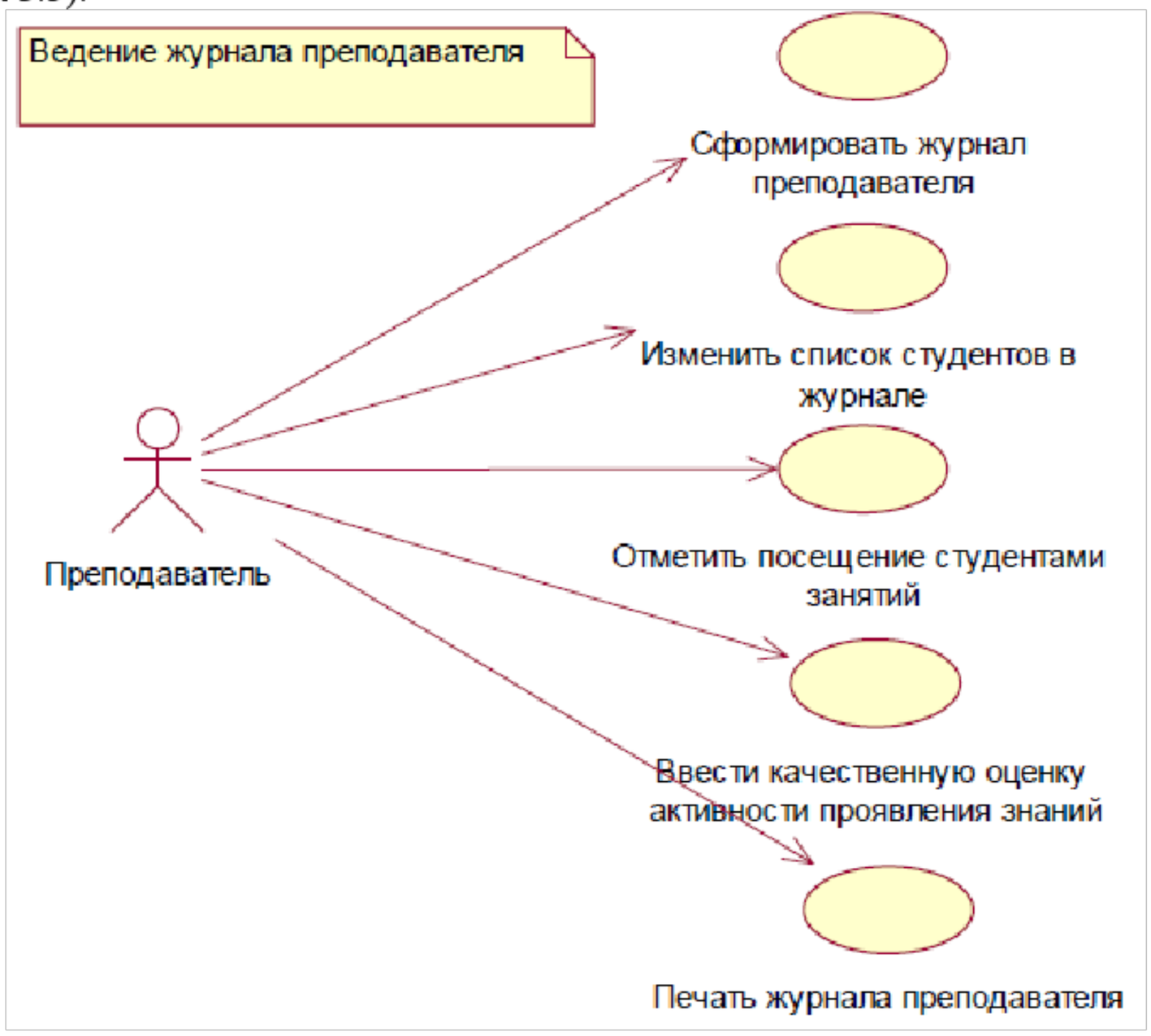


Рисунок 3.5 - Диаграмма вариантов использования

Следующий шаг состоит в уточнении деталей функционального поведения каждого варианта использования. Спецификации вариантов использования состоят из текстовых и графических описаний каждого варианта использования, написанных с позиции пользователя.

3.2.1.2 Спецификация вариантов использования

Определение потока событий. Сердцевиной варианта использования является поток событий. Это текстовое описание операций актанта и различных ответов системы. Поток событий описывает, что, как предполагается, будет делать система в зависимости от поведения актанта. Не обязательно описывать поток в текстовой форме. Для этого можно использовать диаграммы взаимодействия UML, а также другие формальные методы. Главное – обеспечить понимание, так как единственного подхода на все случаи жизни не существует. Однако, как правило, вполне подходит описание на естественном языке.

Поток событий описывает достижение цели варианта использования и предназначен для рассмотрения следующими заинтересованными лицами:

- клиентами, которые одобряют результат;
- пользователями, которые будут работать с системой;
- разработчиками вариантов использования, которые заинтересованы в точном описании желаемого поведения системы;
- рецензентами, которые составляют непредвзятое мнение о системе;
- проектировщиками, которые анализируют варианты использования в поисках классов, объектов и т. п.;
- тестерами, которым нужно создать тестовые примеры;
- менеджером проекта, которому необходимо понимать весь проект в целом;
- составителем технической документации, которому нужно документировать функции системы так, чтобы было понятно пользователю;
- людьми из отдела маркетинга и продаж, которым необходимо понимать функции продукта и объяснять его достоинства остальным.

Альтернативный поток событий. Вариант использования может иметь различные потоки в зависимости от возникающих условий. Иногда эти потоки связаны с выявленными в процессе обработки ошибочными условиями, в других случаях они могут описывать дополнительные

способы обработки конкретных условий.

В нашем примере альтернативный поток событий возникает, когда Жилец удерживает кнопку пульта в нажатом состоянии дольше одной секунды. Нам нужно добавить в вариант использования альтернативный поток.

Выявление пред- и постусловий. Использовать пред- и постусловия нужно только тогда, когда необходимо прояснить поведение, выраженное вариантом использования.

Важно проводить различие между событиями, которые запускают потоки варианта использования, и предусловиями, которые должны быть выполнены до того, как можно будет инициировать вариант использования. Например, предусловием для варианта использования «Управление освещением» является то, что домовладелец (Жилец) должен обеспечить наличие определенного набора осветительных приборов, способных изменять яркость.

Еще одним предусловием является то, что выбранная кнопка пульта должна быть запрограммирована для управления этим набором. (Предполагается, что другие варианты использования описывают, как осуществляются эти предусловия.) Итак, нам нужно сформулировать предусловия.

Постусловия позволяют точно указывать состояние, которое должно быть истинным по окончании варианта использования, даже если использовались альтернативные пути.

Чтобы яркость была такой, как нужно, когда Жилец включает свет в следующий раз, система должна «помнить» уровень яркости, который был установлен для выбранной кнопки пульта после действий по изменению яркости. Следовательно, это постусловие, которое мы должны записать для данного варианта использования.

3.2.1.3 Преимущества

Применение вариантов использования имеет ряд преимуществ по сравнению с традиционным подходом, когда определяются отдельные декларативные программные требования:

- варианты использования относительно легко писать;
- варианты использования пишутся на языке пользователя;
- варианты использования предлагают связные сценарии поведения, которые понятны как пользователю, так и разработчику.

Благодаря описанию «сценариев поведения», содержащимся в языке UML специализированным элементам и нотациям моделирования, варианты использования обеспечивают дополнительные возможности связывать деятельность по разработке требований с проектированием и реализацией.

Графическое представление вариантов использования в UML и их поддержка различными инструментальными средствами моделирования обеспечивают визуализацию связей между вариантами использования, что может содействовать пониманию сложной программной системы.

Сценарий, описанный с помощью варианта использования, может практически без изменений применяется в качестве сценария тестирования во время проверки правильности.

3.2.2 Псевдокод

Псевдокод – это квазиязык программирования; попытка соединить неформальный естественный язык со строгими синтаксическими и управляющими структурами языка программирования. В чистом виде псевдокод состоит из комбинации следующих элементов.

- Императивных предложений с одним глаголом и одним объектом.
- Ограниченного множества (как правило, не более 40–50) «ориентированных на действия» глаголов, из которых должны конструироваться предложения.
- Решений, представленных формальной структурой IF-ELSE-ENDIF.
- Итеративных действий, представленных структурами DO-WHILE и FOR-NEXT.

На рисунке 3.6 представлен пример спецификации с помощью псевдокода алгоритма вычисления отложенного дохода от услуг в течение данного месяца в бизнес-приложении.

```
Set Sum(X)=0
for каждого клиента x
    if клиент оплатил услуги вперед
        and ((Текуший месяц)>=(2 мес. после даты приобретения)) and ((Текуший
        месяц)<=(14 мес. после даты приобретения))
    then Sum(X) = Sum(x) + (сумма, заплаченная клиентом)/12
```

Рисунок 3.6 - Пример спецификации с использованием псевдокода

Фрагменты текста на псевдокоде расположены уступами; такой формат используется для того, чтобы выделить логические блоки. Сочетание синтаксических ограничений, формата и разбивки существенно уменьшает неоднозначность требования, которое в противном случае было бы очень сложным и расплывчатым. В то же время такое представление требования вполне понятно для человека, не являющегося программистом. Не нужно быть выдающимся ученым, чтобы понимать псевдокод, и нет необходимости знать C++ или Java.

3.2.3 Конечные автоматы

Представление в виде конечного автомата описывает возможные жизненные циклы объекта и состоит из состояний, соединенных переходами.

Каждое состояние – это такой период жизненного цикла объекта, когда он удовлетворяет определенным условиям. Некоторое событие может привести к переходу, в результате которого объект окажется в новом состоянии.

При переходе может выполняться действие, предписанное данному переходу.

Представление в виде конечного автомата изображается на диаграммах состояний и переходов.

На рисунке 3.7 приведен пример диаграммы состояний и переходов для объекта «Заказ».

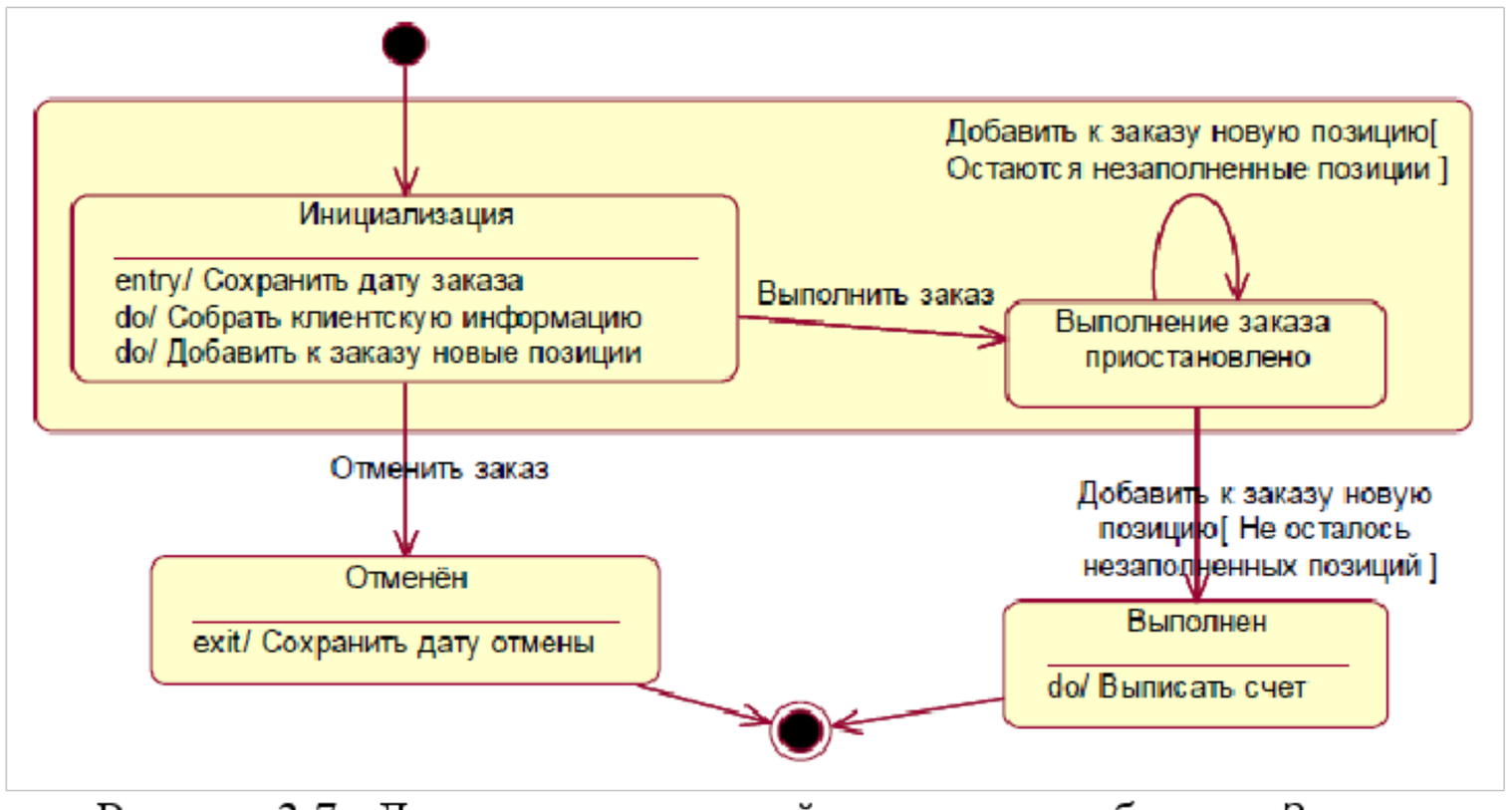


Рисунок 3.7 - Диаграмма состояний и переходов объекта «Заказ»

3.2.4 Графические деревья решений

Дерево решений применяется для отображения информации в виде графа. На рисунке 3.8 показано дерево решений, используемое для описания последовательности действий.

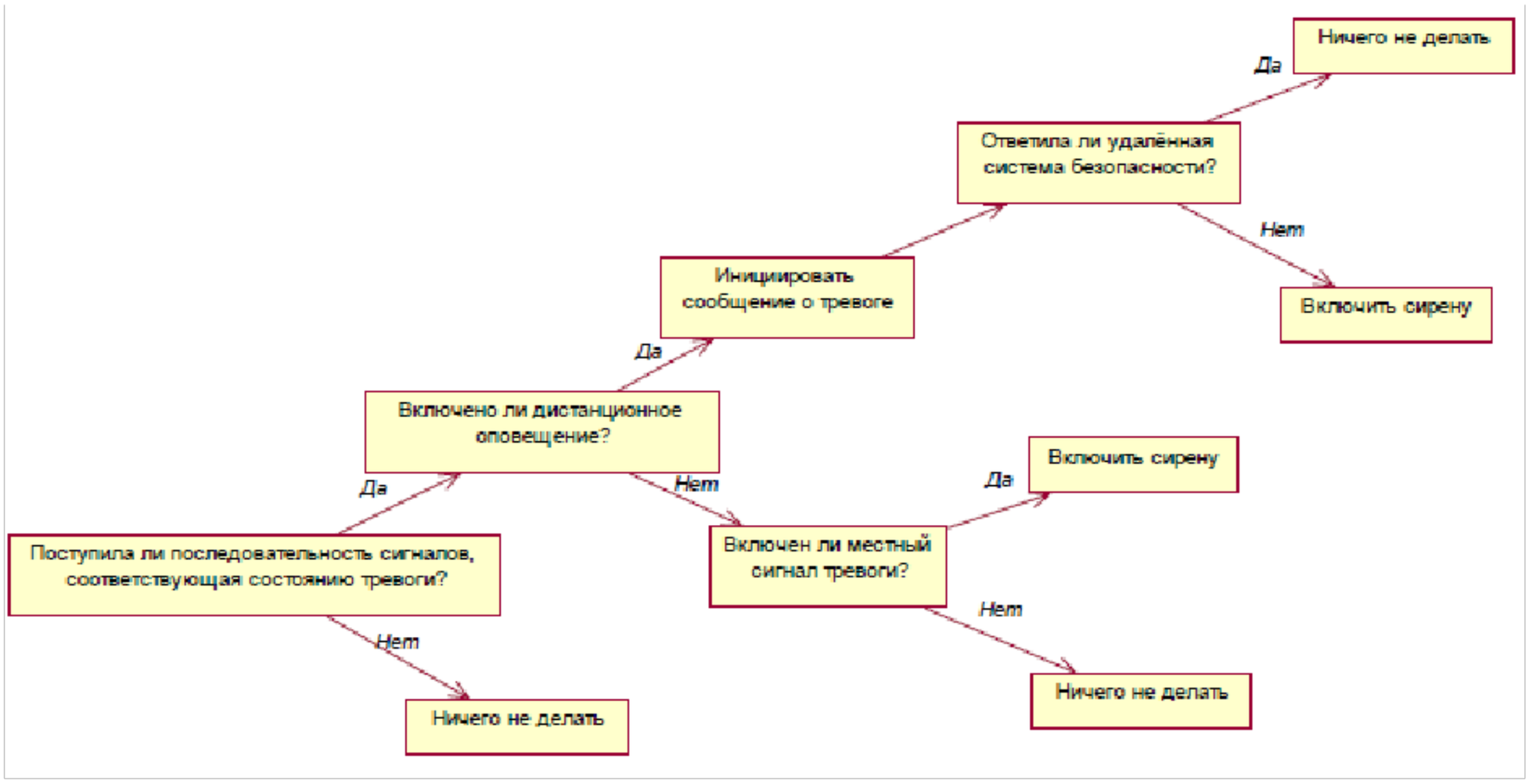


Рисунок 3.8 - Графическое дерево решений

3.2.5 Диаграммы деятельности

Деятельность представляет собой поток работ или выполнение операции. Представление деятельности отображает как последовательные, так и параллельные виды деятельности. Изображаются такие модели на диаграммах деятельности.

На рисунке 3.9 изображена диаграмма, которая описывает деятельности покупателя в Интернет-магазине. Здесь представлены две точки ветвления – для выбора способа поиска товара и для принятия решения о покупке. Присутствуют три линейки синхронизации: верхняя – отражает разделение на два параллельных процесса, средняя отражает и разделение, и слияние процессов, а нижняя – только слияние процессов. Дополнительно на этой диаграмме показаны две дорожки – дорожка покупателя и дорожка магазина, которые разделены вертикальной линией. Каждая дорожка имеет имя и фиксирует область деятельности конкретного лица, обозначая зону его ответственности.

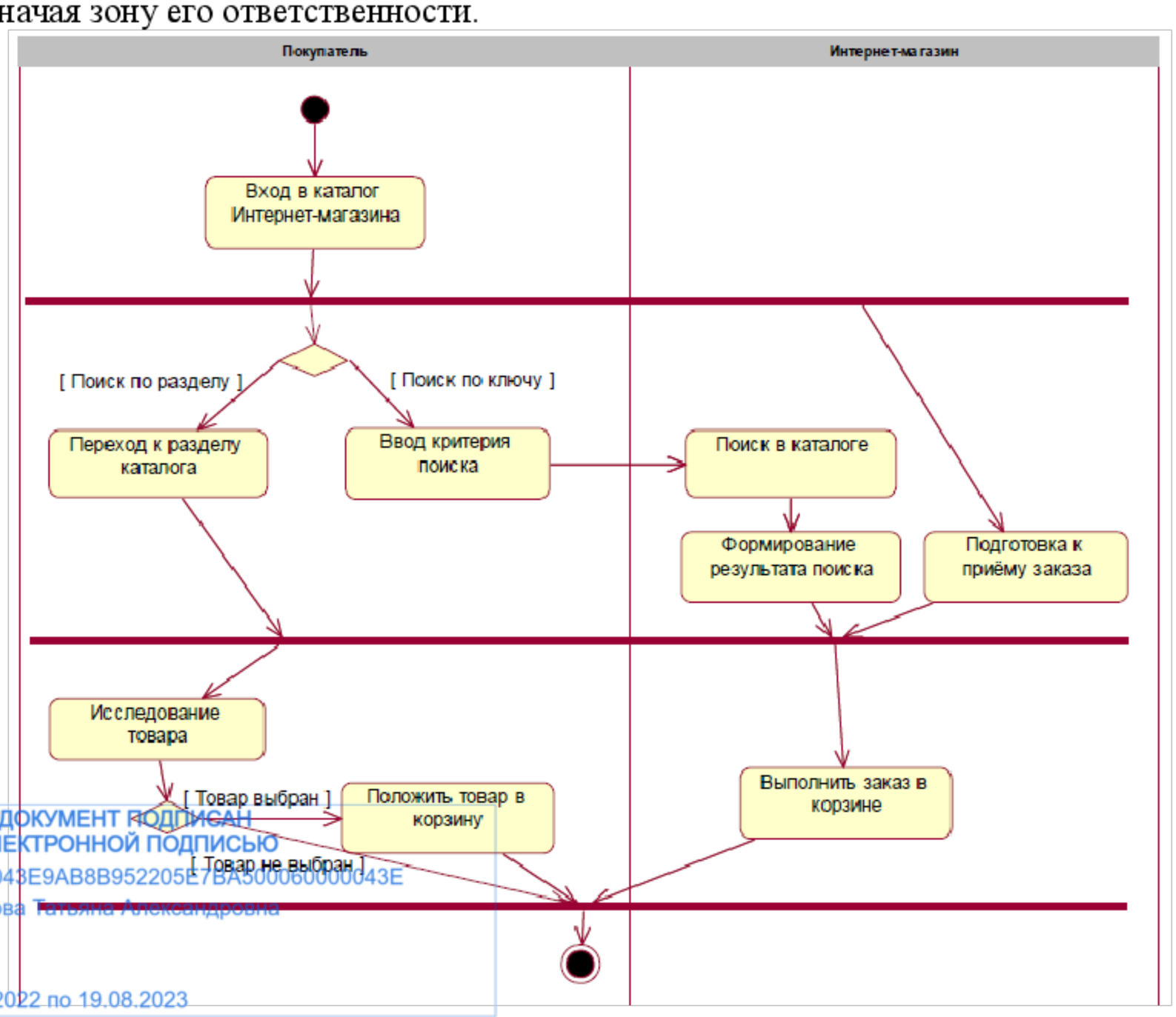


Рисунок 3.9 - Диаграмма деятельности покупателя в Интернет-магазине

Диаграмма деятельности отражает реальные потоки работ в человеческой организации. Такое бизнес-моделирование и есть основное ее назначение. С таким же успехом ее можно использовать и при моделировании работ программного приложения.

3.3 Техническое задание (ГОСТ34.602–89)

Настоящий стандарт распространяется на автоматизированные системы (АС) для автоматизации различных видов деятельности (управление, проектирование, исследование и т. п.), включая их сочетания, и устанавливает состав, содержание, правила оформления документа «Техническое задание на создание (развитие или модернизацию) системы».

3.3.1 Общие сведения

В этот раздел включают: полное наименование системы и ее условное обозначение; шифр темы или шифр (номер) договора; наименование предприятий (объединений) разработчика и заказчика (пользователя) системы и их реквизиты; перечень документов, на основании которых создается система, кем и когда утверждены эти документы; плановые сроки начала и окончания работы по созданию системы; сведения об источниках и порядке финансирования работ; порядок оформления и предъявления заказчику результатов работ по созданию системы (ее частей), по изготовлению и наладке отдельных средств (технических, программных, информационных) и программно-технических (программно-методических) комплексов системы.

3.3.2 Назначение и цели создания (развития) системы

3.3.2.1 Назначение системы

Здесь указывают вид автоматизируемой деятельности (управление, проектирование и т. п.) и перечень объектов автоматизации, на которых предполагается ее использовать. Для АСУ дополнительно указывают перечень автоматизируемых органов (пунктов) управления и управляемых объектов.

3.3.2.2 Цели создания системы

Приводят наименования и требуемые значения технических, технологических, производственно-экономических или других показателей объекта автоматизации, которые должны быть достигнуты в результате создания АС, и указывают критерии оценки достижения целей создания системы.

3.3.3 Характеристики объекта автоматизации

В данном разделе приводят: краткие сведения об объекте автоматизации или ссылки на документы, содержащие такую информацию; сведения об условиях эксплуатации объекта автоматизация и характеристиках окружающей среды.

Примечание. Для САПР дополнительно приводят основные параметры и характеристики объектов проектирования.

3.3.4 Требования к системе

Состав требований к системе, включаемых в данный раздел ТЗ на АС, устанавливают в зависимости от вида, назначения, специфических особенностей и условий функционирования конкретной системы. В каждом подразделе приводят ссылки на действующие НТД, определяющие требования к системам соответствующего вида.

3.3.4.1 Требования к системе в целом

Требования к структуре и функционированию системы. Здесь приводят: перечень

подсистем, их назначение и основные характеристики, требования к числу уровней иерархии и степени централизации системы; требования к способам и средствам связи для информационного обмена между компонентами системы; требования к характеристикам взаимосвязей создаваемой системы со смежными системами, требования к ее совместимости, в том числе указания о способах обмена информацией (автоматически, пересылкой документов, по телефону и т. п.); требования к режимам функциони- рования системы; требования по диагностированию системы; перспективы развития, модернизации системы.

Требования к численности и квалификации персонала на АС. Приводят: требования к численности персонала (пользователей) АС; требования к квалификации персонала, порядку его подготовки и контроля знаний и навыков; требуемый режим работы персонала АС.

Требования к показателям назначения АС. Приводят значения параметров, характеризующих степень соответствия системы ее назначению.

Для АСУ указывают: степень приспособляемости системы к изменению процессов, к отклонениям параметров объекта управления; номер методов управления; допустимые пределы модернизации и развития системы; вероятностно-временные характеристики, при которых сохраняется целевое назначение системы.

Требования к надежности. Включают: состав и количественные значения показателей надежности для системы в целом или ее подсистем; перечень аварийных ситуаций, по которым должны быть регламентированы требования к надежности, и значения соответствующих показателей; требования к надежности технических средств и программного обеспечения; требования к методам оценки и контроля показателей надежности на разных стадиях создания системы в соответствии с действующими нормативно-техническими документами.

Требования по безопасности. Включают требования по обеспечению безопасности при монтаже, наладке, эксплуатации, обслуживании и ремонте технических средств системы (защита от воздействий электрического тока, электромагнитных полей, акустических шумов и т. п.), по допустимым уровням освещенности, вибрационных и шумовых нагрузок.

Требования по эргономике и технической эстетике. Включают показатели АС, задающие необходимое качество взаимодействия человека с машиной и комфортность условий работы персонала.

Требования к эксплуатации, техническому обслуживанию, ремонту и хранению. Включают: условия и регламент (режим) эксплуатации, которые должны обеспечивать использование технических средств (ТС) системы с заданными техническими показателями, в том числе виды и периодичность обслуживания ТС системы или допустимость работы без обслуживания; предварительные требования к допустимым площадям для размещения персонала и ТС системы, к параметрам сетей энергоснабжения и т. п.; требования по количеству, квалификации обслуживающего персонала и режимам его работы; требования к составу, размещению и условиям хранения комплекта запасных изделий и приборов; требования к регламенту обслуживания.

Требования к защите информации от несанкционированного доступа. Включают требования, установленные в НТД, действующей в отрасли (ведомстве) заказчика.

Требования по сохранности информации. Приводят перечень событий: аварий, отказов технических средств (в том числе потеря питания) и т. п., при которых должна быть обеспечена сохранность информации в системе.

Требования к средствам защиты от внешних воздействий. Приводят: требования к радиоэлектронной защите средств АС; требования по стойкости, устойчивости и прочности к внешним воздействиям (среде применения).

Требования к патентной чистоте. Указывают перечень стран, в отношении которых должна быть обеспечена патентная чистота системы и ее частей.

Требования к стандартизации и унификации. Включают: показатели, устанавливающие требуемую степень использования стандартных, унифицированных методов реализации функций (задач) системы, поставляемых программных средств, типовых математических методов и моделей, типовых проектных решений, унифицированных форм управленческих документов, установленных ГОСТ 6.10.1, общесоюзных классификаторов технико-экономической информации и классификаторов других категорий в соответствии с областью их применения, требования к использованию типовых автоматизированных рабочих мест, компонентов и комплексов.

Дополнительные требования. Включают: требования к оснащению системы

устройствами для обучения персонала (тренажерами, другими устройствами аналогичного назначения) и документацией на них; требования к сервисной аппаратуре, стендам для проверки элементов системы; требования к системе, связанные с особыми условиями эксплуатации; специальные требования по усмотрению разработчика или заказчика системы.

3.3.4.2 Требования к функциям (задачам)

Здесь приводят: по каждой подсистеме перечень функций, задач или их комплексов (в том числе обеспечивающих взаимодействие частей системы), подлежащих автоматизации; при создании системы в две или более очереди – перечень функциональных подсистем, отдельных функций или задач, вводимых в действие в 1-й и последующих очередях; временной регламент реализации каждой функции, задачи (или комплекса задач); требования к качеству реализации каждой функции (задачи или комплекса задач), к форме представления выходной информации, характеристики необходимой точности и времени выполнения, требования одновременности выполнения группы функций, достоверности выдачи результатов; перечень и критерии отказов для каждой функции, по которой задаются требования по надежности.

3.3.4.3 Требования к видам обеспечения

В зависимости от вида системы приводят требования к математическому, информационному, лингвистическому, программному, техническому, метрологическому, организационному, методическому и другие видам обеспечения системы.

Требования к математическому обеспечению системы. Приводят требования к составу, области применения (ограничения) и способам использования в системе математических методов и моделей, типовых алгоритмов и алгоритмов, подлежащих разработке.

Требования к информационному обеспечению. Приводят требования: к составу, структуре и способам организации данных в системе; к информационному обмену между компонентами системы; к информационной совместимости со смежными системами; по использованию общесоюзных и зарегистрированных республиканских, отраслевых классификаторов, унифи- цированных документов и классификаторов, действующих на данном предприятии; по применению систем управления базами данных; к структуре процесса сбора, обработки, передачи данных в системе и представлению данных; к защите данных от разрушений при авариях и сбоях в электропитании системы; к контролю, хранению, обновлению и восстановлению данных; к процедуре придания юридической силы документам, продуцируемым техническими средствами АС (в соответствии с ГОСТ 6.10.4).

Требования к лингвистическому обеспечению. Приводят требования к применению в системе языков программирования высокого уровня, языков взаимодействия пользователей и технических средств системы, а также требования к кодированию и декодированию данных, к языкам ввода-вывода данных, языкам манипулирования данными, средствам описания предметной области (объекта автоматизации), к способам организации диалога.

Требования к программному обеспечению. Приводят перечень покупных программных средств. А также требования: к независимости программных средств от используемых комплексов технических средств и операционной среды; к качеству программных средств, а также к способам его обеспечения и контроля; по необходимости согласования вновь разрабатываемых программных средств с фондом алгоритмов и программ.

Требования к техническому обеспечению. Приводят требования: к видам технических средств, в том числе к видам комплексов технических средств, программно-технических комплексов и других комплектующих изделий, допустимых к использованию в системе; к функциональным, конструктивным и эксплуатационным характеристикам средств технического обеспечения системы.

Требования к метрологическому обеспечению. Приводят: предварительный перечень измерительных каналов; требования к точности измерений параметров и (или) к метрологическим характеристикам измерительных каналов; требования к метрологической совместимости технических средств системы; перечень управляющих и вычислительных каналов системы, для которых необходимо оценивать точностные характеристики; требования к метрологическому обеспечению технических и программных средств, входящих в состав измерительных каналов системы, средств, встроенного контроля, метрологической пригодности

измерительных каналов и средств измерений, используемых при наладке и испытаниях системы; вид метрологической аттестации (государственная или ведомственная) с указанием порядка ее выполнения и организаций, проводящих аттестацию.

Требования к организационному обеспечению. Приводят: к структуре и функциям подразделений, участвующих в функционировании системы или обеспечивающих эксплуатацию; к организации функционирования системы и порядку взаимодействия персонала АС и персонала объекта автоматизации; к защите от ошибочных действий персонала системы.

Требования к методическому обеспечению САПР. Приводят требования к составу нормативно-технической документации системы (перечень применяемых при ее функционировании стандартов, нормативов, методик и т. п.).

3.3.5 Состав и содержание работ по созданию (развитию) системы

Содержаться перечень стадий и этапов работ по созданию системы в соответствии с ГОСТ 24.601, сроки их выполнения, перечень организаций – исполнителей работ, ссылки на документы, подтверждающие согласие этих организаций на участие в создании системы, или запись, определяющую ответственного (заказчик или разработчик) за проведение этих работ.

Кроме того приводят:

- перечень документов по ГОСТ 34.201, предъявляемых по окончании соответствующих стадий и этапов работ;
- вид и порядок проведения экспертизы технической документации (стадия, этап, объем проверяемой документации, организация-эксперт);
- программу работ, направленных на обеспечение требуемого уровня надежности разрабатываемой системы (при необходимости);
- перечень работ по метрологическому обеспечению на всех стадиях создания системы с указанием их сроков выполнения и организаций-исполнителей (при необходимости).

3.3.6 Порядок контроля и приемки системы

Указывают: виды, состав, объем и методы испытаний системы и ее составных частей (виды испытаний в соответствии с действующими нормами, распространяющимися на разрабатываемую систему); общие требования к приемке работ по стадиям (перечень участвующих предприятий и организаций, место и сроки проведения), порядок согласования и утверждения приемочной документации; статус приемочной комиссии (государственная, межведомственная, ведомственная).

3.3.7 Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие

Здесь необходимо привести перечень основных мероприятий и их исполнителей, которые следует выполнить при подготовке объекта автоматизации к вводу АС в действие.

В перечень основных мероприятий включают: приведение поступающей в систему информации (в соответствии с требованиями к информационному и лингвистическому обеспечению) к виду, пригодному для обработки с помощью ЭВМ; изменения, которые необходимо осуществить в объекте автоматизации; создание условий функционирования объекта автоматизации, при которых гарантируется соответствие создаваемой системы требованиям, содержащимся в ТЗ; создание необходимых для функционирования системы подразделений и служб; сроки и порядок комплектования штатов и обучения персонала.

Например, для АСУ приводят: изменения применяемых методов управления; создание условий для работы компонентов АСУ, при которых гарантируется соответствие системы требованиям, содержащимся в ТЗ.

3.3.8 Требования к документированию

Приводят: согласованный разработчиком и заказчиком системы перечень подлежащих разработке комплектов и видов документов, соответствующих требованиям ГОСТ 34.201 и НТД отрасли заказчика; перечень документов, выпускаемых на машинных носителях; требования

к микрофильмированию документации; требования по документированию комплектующих элементов межотраслевого применения в соответствии с требованиями ЕСКД и ЕСПД; при отсутствии государственных стандартов, определяющих требования к документированию элементов системы, дополнительно включают требования к составу и содержанию таких документов.

3.3.9 Источники разработки

Должны быть перечислены документы и информационные материалы (технико-экономическое обоснование, отчеты о законченных научно-исследовательских работах, информационные материалы на отечественные, зарубежные системы-аналоги и др.), на основании которых разрабатывалось ТЗ и которые должны быть использованы при создании системы.

План конспекта:

1. Введение в системный анализ.
2. Анализ проблемы и моделирование предметной области с использованием системного подхода.
3. Методология ARIS.
4. Стандарты IDEF0 – IDEF3.
5. Методы определения требований.
6. Формализация требований.
7. Техническое задание (ГОСТ 34.602–89).

Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-3	1-3	1-3	1-2

Тема самостоятельного изучения № 4
Архитектуры программных систем.

Вид деятельности студентов: самостоятельное изучение литературы
Итоговый продукт самостоятельной работы: конспект
Средства и технологии оценки: собеседование
Краткие теоретические сведения

4 АРХИТЕКТУРЫ ПРОГРАММНЫХ СИСТЕМ

4.1 Планирование архитектуры

Современные методы разработки программного обеспечения предполагают обратную связь между всеми действующими лицами, от проектировщика до аналитика. Все эти лица являются участниками процесса создания архитектуры программной системы. Под архитектурой системы будем понимать структуру компонентов программной системы, взаимосвязи, а также принципы и нормы их проектирования и развития во времени. Прежде чем начать изучение процесса планирования архитектуры, необходимо познакомиться с понятием архитектурно-экономического цикла (АЭЦ).

4.1.1 Архитектурно-экономический цикл

Взаимоотношения между производственными задачами, требования к продукту, опыт архитектора, архитектуры и созданные системы образуют цикл с цепями обратной связи. Упомянутые цепи обратной связи изображены на рисунке 4.1. Частично обратная связь поступает от самой архитектуры, частично – от построенной на ее основе системы.

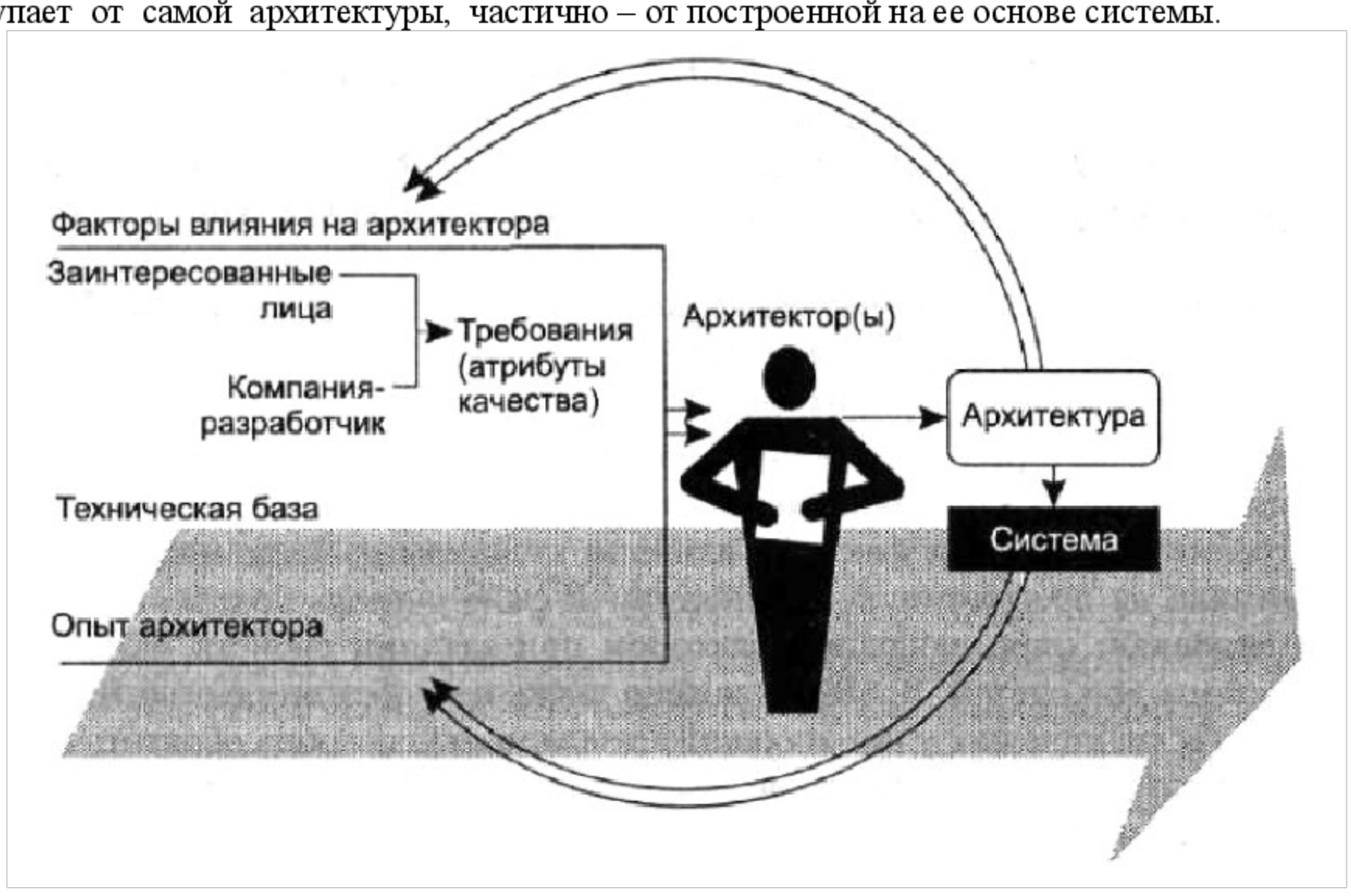


Рисунок 4.1 Архитектурно-экономический цикл

Цикл выглядит следующим образом.

1. Архитектура влияет на структуру компании-разработчика. Архитектура обусловливает структуру системы; в частности (в этом мы сможем убедиться), она устанавливает набор блоков программного обеспечения, которое надлежит реализовать (или обеспечить их наличие другим путем), а затем интегрировать в рамках системы. Эти блоки составляют основу разработки структуры проекта. Группы разработчиков укомплектовываются именно по блокам, операции в рамках процессов разработки, тестирования и интеграции также выполняются в отношении блоков. Согласно графикам и бюджетам, ресурсы выделяются частями в расчете на отдельные блоки. Если компания наработала опыт конструирования семейств сходных систем, она будет вкладывать средства в повышение профессионального уровня участников сформированных по блокам групп разработчиков. Следовательно, группы встраиваются в структуру организации. Такой представляется обратная связь от архитектуры к компании-разработчику.

2. Архитектура способна оказывать воздействие на задачи компании-разработчика. Сконструированная на ее основе успешная система предоставляет компании возможность укрепиться в данном сегменте рынка. Такая архитектура предусматривает дальнейшее эффективное производство и размещение сходных систем, вследствие чего компания может откорректировать свои задачи и, воспользовавшись новым преимуществом, занять рыночную

нишу. Так выглядит обратная связь от системы к компании-разработчику и конструируемым ею системам.

3. Архитектура может оказывать воздействие на требования, выдвигаемые заказчиком относительно следующей системы, – ее (если она основана на той же архитектуре, что и предыдущая) он может получить в более надеж ном варианте, быстрее и экономичнее, чем в том случае, если бы она конструировалась с чистого листа. Возможно, заказчик откажется от некоторых требований в пользу повышения экономичности. Готовые программные продукты несколько изменили требования, предъявляемые заказчиками, – не предназначенные для удовлетворения индивидуальных потребностей, они недороги и отличаются высоким качеством. На заказчиков, не слишком гибких по части своих требований, аналогичное воздействие оказывают линейки продуктов.

4. Процесс конструирования систем пополняет опыт архитектора, который он может применить при работе над последующими архитектурами, и, соответственно, расширяет базу опыта компании. Успех системы, построенной на основе инструментальных магистралей, .NET или инкапсулированных конечных автоматов, стимулирует построение аналогичных систем в дальнейшем. С другой стороны, неудачные варианты архитектуры редко используются повторно.

5. Иногда оказать сильное воздействие и даже внести изменения в культуру программной инженерии (техническую базу, в рамках которой обучаются и работают конструкторы) способны отдельные системы. Такой эффект на индустрию в 1960-х и начале 1970-х гг. оказали первые реляционные базы данных, генераторы компиляторов и табличные операционные системы; в 1980-х – первые электронные таблицы и системы управления окнами. В 1990-х гг. в качестве такого рода катализатора выступила Всемирная паутина. Подобные инновации всегда находят отражение в последующих системах.

Из этих и некоторых других механизмах обратной связи и образуется архитектурно-экономический цикл. На рисунке 4.1 изображены факторы влияния культуры и экономики компании-разработчика на программную архитектуру. В свою очередь архитектура оказывается основным определяющим фактором при задании свойств разрабатываемой системы или систем.

4.1.2 Программный процесс и архитектурно-экономический цикл

Программным процессом (software process) называются действия по организации, нормированию и управлению разработкой программного обеспечения. Ниже приведен перечень операций, направленных на создание программной архитектуры, ее применение для реализации проектного решения, а впоследствии – на реализацию или управление развитием целевой системы или приложения:

- создание экономической модели системы;
- выявление требований;
- создание новой или выбор существующей архитектуры;
- документирование и распространение сведений об архитектуре;
- анализ или оценка архитектуры;
- реализация системы на основе архитектуры;
- проверка соответствия реализации архитектуре.

4.1.2.1 Этапы разработки архитектуры

Как следует из структуры АЭЦ, между различными этапами разработки архитектуры существуют развернутые отношения обратной связи

Создание экономической модели системы. Создание экономической модели не ограничивается оценкой потребности в системе на рынке. Этот этап исполняет существенную роль в контексте создания и сужения требований. Сколько должен стоить продукт? Каков целевой сегмент рынка? Насколько быстро продукт должен выйти на рынок? Должен ли он взаимодействовать с другими системами? Есть ли какие-нибудь системные ограничения, в рамках которых он должен существовать? Все эти вопросы решаются с привлечением архитектора. Одного его, конечно, недостаточно, однако если при создании экономической модели консультации с архитектором не проводились, то возможность достижения коммерческих задач становится проблематичной.

Выявление требований. Способов узнать, чего же, наконец, хотят заинтересованные лица, множество. В частности, в рамках объектно-ориентированного анализа для фиксации

требований используются сценарии, или элементы Use Case. Для системы с повышенными требованиями к безопасности применяются более строгие методики – например, модели конечных автоматов и языки формальных спецификаций.

Применительно к конструируемой системе необходимо принять центральное, основополагающее решение – насколько в ней будут отражены другие, уже сконструированные системы. Поскольку в сегодняшних условиях найти систему, не имеющую сходств с другими системами, весьма непросто, методики выявления требований предполагают знание характеристик предшествующих систем.

Еще один способ выявления требований подразумевает моделирование. Опытные системы помогают моделировать нужное поведение, проектировать пользовательские интерфейсы и проводить анализ потребления ресурсов. Таким образом, в глазах заинтересованных лиц система становится «реальной», а процесс принятия решений по проектированию системы и ее пользовательского интерфейса значительно ускоряется.

Вне зависимости от методики выявления требований, желаемые атрибуты качества конструируемой системы обусловливают ее конечный вид. Для обеспечения отдельных атрибутов качества архитекторами уже давно применяются те или иные тактики. Архитектурные решения компромиссны, однако при специфицировании требований не все эти компромиссы очевидны. Со всей ясностью они проявляются только после создания архитектуры; тогда же принимаются решения относительно сортировки требований в соответствии с приоритетами.

Создание или выбор архитектуры. Фредерик Брукс (Fred Brooks) в своей знаменитой книге «Мифический человеко-месяц» красноречиво и убедительно доказывает, что основным условием стабильного проектирования системы является соблюдение концептуальной целостности, а она может проявиться лишь в узком кругу людей, совместно работающих над проектированием ее архитектуры.

Распространение сведений об архитектуре. Для того чтобы архитектура действительно стала основой проекта, ее суть необходимо четко и недвусмысленно донести до всех заинтересованных лиц. Разработчики должны понимать, что от них требуется, тестировщики должны осознавать структуру своих задач, менеджмент должен знать график и т. д. Для того чтобы этой цели можно было добиться, документирование архитектуры должно быть информативным, ясным и понятным людям различных профессий.

Анализ или оценка архитектуры. В процессе проектирования всегда рассматривается множество вариантов проекта. Некоторые из них забраковываются сразу. Из числа остальных в конечном итоге отбирается наиболее подходящий. Одна из глобальных задач, стоящих перед любым архитектором, заключается именно в том, чтобы сделать этот выбор рационально.

Оценить архитектуру на предмет атрибутов качества, которые она обеспечивает, совершенно необходимо – без этого нельзя быть уверенным в том, что конечная система сможет удовлетворить все потребности заинтересованных лиц. Все большее распространение получают методики анализа, ориентированные на оценку сообщаемых системе архитектурой атрибутов качества. Сценарные методики обеспечивают наиболее универсальную и эффективную оценку архитектуры. Самая зрелая методическая база характерна для метода анализа компромиссных архитектурных решений (Architecture Tradeoff Analysis Method, ATAM); метод анализа стоимости и эффективности (Cost Benefit Analysis Method, CBAM), с другой стороны, предусматривает крайне ценную возможность увязки архитектурных решений с их экономическим содержанием.

4.1.3 Суть программной архитектуры

Программная архитектура программы или вычислительной системы – это структура ее структур, т. е. изложение ее программных элементов, их внешних свойств и установленных между ними отношений.

Внешними (externally visible) свойствами называются те предположения, которые сторонние элементы могут выдвигать в отношении данного элемента, – в частности, они касаются предоставляемых элементом услуг, рабочих характеристик, устранения неисправностей, совместного использования ресурсов и т. д. Проанализируем представленное определение более подробно.

Во-первых, архитектура определяет программные элементы. В составе архитектуры

приводится информация о взаимоотношениях элементов. Собственно говоря, все, что архитектура сообщает нам об элементах, ограничивается информацией об их взаимодействии. Итак, архитектура – это, в первую очередь, абстракция системы, в которой отсутствует информация об элементах, не имеющая отношения к тому, как они используют, используются, соотносятся или взаимодействуют с другими элементами. Практически во всех современных системах взаимодействие между элементами осуществляется посредством интерфейсов, которые поддерживают деление информации об элементах на публичную и приватную части. Так вот, архитектура имеет дело только с публичной частью; приватные детали элементов, те, что относятся исключительно к их внутренней реализации, в состав архитектуры не входят.

Во-вторых, из вышеприведенного определения ясно, что в состав любой системы может входить и входит целый ряд структур, а следовательно, ни одной отдельно взятой структуры, которую можно было бы уверенно назвать архитектурой, не существует. В частности, все нетривиальные проекты делятся на блоки реализации; между этими блоками распределяются некоторые обязанности, и они же в большинстве случаев выступают в качестве базы для разделения задач между группами программистов. В таких элементах наряду с программами и данными, доступными для вызова или обращения со стороны других программных средств и блоков реализации, содержатся приватные программы и данные. В крупных проектах эти элементы почти всегда разделяются на мелкие части, а обязанности по работе с ними распределяются между несколькими мелкими группами разработчиков. Довольно часто описания систем составляются при помощи таких структур. Ориентированные на разделение функций системы между различными группами исполнителей реализации, они отличаются чрезмерной статичностью.

Другие структуры в большей степени ориентируются на взаимодействие элементов в период прогона с целью исполнения функции системы. Представим, что систему предполагается сконструировать в виде ряда паралельных процессов. В этом случае часто применяется другая структура, включающая процессы периода прогона, программы из вышеописанных блоков реализации, совместно формирующие отдельные процессы, а также установленные между этими процессами отношения синхронизации.

Можем ли мы взять любую из этих структур в отдельности и назвать ее архитектурой? Нет, не можем – и это несмотря на то, что все они содержат архитектурные сведения. В состав любой архитектуры эти структуры входят наравне со многими другими. В этой связи, очевидно, что, поскольку в составе архитектуры может содержаться несколько структур, в ней должны присутствовать несколько элементов (например, блок реализации и процессы), несколько вариантов взаимодействия между элементами (например, разбиение на составляющие и синхронизация) и даже несколько контекстов (например, время разработки и время прогона). Представленное определение не устанавливает сущность архитектурных элементов и отношений. Является ли программный элемент объектом? процессом? библиотекой? базой данных? коммерческим изделием? Он может быть чем угодно, причем возможности не ограничиваются вышеприведенными вариантами.

В-третьих, согласно нашему определению, программная архитектура есть у любой вычислительной системы с программным обеспечением. Связано это с тем, что любую систему можно представить как совокупность ее элементов и установленных между ними отношений. В простейшем случае система сама по себе является элементом, не представляющим, вероятно, никакого интереса и бесполезным, однако, тем не менее, согласующимся с понятием «архитектура». То обстоятельство, что архитектура есть у каждой системы, совершенно не означает, что она является общеизвестной. Кто знает, быть может, специалистов, спроектировавших систему, уже не найти, документация тоже куда-то исчезла (или ее никогда не существовало), исходный код потерян (или не поставлялся), а остался лишь исполняемый двоичный код. Именно в этом случае различие между архитектурой системы и ее представлением становится очевидным. К сожалению, архитектура иногда существует самостоятельно, без описаний и спецификаций; в этой связи существенную важность приобретают документирование и реконструкция архитектуры.

В-четвертых, если поведение отдельно взятого элемента можно зафиксировать или выявить с точки зрения других элементов, то это поведение входит в состав архитектуры. Именно оно позволяет элементам взаимодействовать друг с другом, а взаимодействие, как известно, отражается в архитектуре в обязательном порядке. Этим, помимо прочего, объясняется, почему схемы из прямоугольников и линий, выдаваемые за архитектуры, таковыми

не являются. Это не более чем рисунки; самое лучшее, что о них можно сказать, – это то, что они намекают на существование более четкой информации относительно фактических функций обозначенных элементов. Взглянув на наименования изображенных на подобной схеме прямоугольников (например, на них написано «база данных», «пользовательский интерфейс», «ис- полняемый файл» и т. д.), читатель хотя бы получит представление о функциональности и поведении соответствующих элементов.

Наконец, в представленном определении не отражено качество архитектуры системы, иначе говоря, никаких утверждений относительно перспектив соответствия системы установленным требованиям к поведению, производительности и жизненному циклу в ней нет. Поскольку метод проб и ошибок (предполагающий произвольный выбор архитектуры и последующее конструирование на ее основе системы под лозунгом «Будь что будет») в качестве оптимального способа выбора архитектуры для системы нас совсем не устраивает, то далее рассмотрим вопросы оценки архитектуры и архитектурного проектирования.

4.1.3.1 Архитектурные образцы, эталонные модели и эталонные варианты архитектуры

В промежутке между схемами из прямоугольников и линий – простейшими «набросками» архитектур – и комплексными архитектурами, укомплектованными всей необходимой информацией о системах, существуют многочисленные переходные этапы. Каждый такой этап есть результат принятия ряда архитектурных решений, совокупность архитектурных альтернатив. Некоторые из них сами по себе имеют определенную ценность. Прежде чем переходить к анализу архитектурных структур, рассмотрим три проме- жуточных этапа.

1. Архитектурный образец – это описание типов элементов и отношений и изложение ряда ограничений на их использование. Образец имеет смысл рассматривать как совокупность ограничений, накладываемых на архитектуру, – конкретнее, на типы элементов и образцы их взаимодействия; на основе этих ограничений складывается ряд или семейство соответствующих им вариантов архитектуры. К примеру, одним из общеупотребительных архитектурных образцов является клиент-сервер. Клиент и сервер – это два типа элементов; их взаимодействие описывается посредством протокола, при помощи которого сервер взаимодействует со всеми своими клиентами. Термин «клиент-сервер» по смыслу лишь предполагает множественность клиентов; конкретные клиенты не перечисляются, и речи о том, какая функциональность, помимо реализации протоколов, характерна для клиентов или для сервера, не идет. Согласно этому (неформальному) определению, образцу «клиент-сервер» соответствует бесчисленное количество различных вариантов архитектуры, причем все они чем-то отличаются друг от друга. Несмотря на то, что архитектурный образец не является архитектурой, он все же содержит весьма полезный образ системы – он накладывает на архитектуру, а следовательно, и на систему, полезные ограничения.

У образцов есть один крайне полезный аспект – дело в том, что они демонстрируют известные атрибуты качества. Именно поэтому архитекторы выбирают образцы не наугад, а исходя из определенных соображений. Некоторые образцы содержат известные решения проблем, связанных с производительностью, другие предназначаются для систем с высокими требованиями к безопасности, третьи успешно реализуются в системах с высокой готовностью. Во многих случаях выбор архитектурного образца оказывается первым существенным решением архитектора.

Синонимичным архитектурному образцу является общеупотребительный термин «архитектурный стиль» (architectural style).

Эталонная модель – это разделение между отдельными блоками функциональных возможностей и потоков данных. Эталонной моделью называется стандартная декомпозиция известной проблемы на части, которые, взаимодействуя, способны ее разрешить. Поскольку эталонные модели имеют происхождение в опыте, их наличие характерно только для сформировавшихся предметных областей. Вы можете назвать стандартные элементы компилятора или системы управления базами данных? А в общих словах объяснить, как эти элементы сообща решают свою общую задачу? Если можете, значит, вы знакомы с эталонной моделью этих приложений.

Эталонная архитектура – это эталонная модель, отображенная на программные элементы (которые сообща реализуют функциональность, определенную в эталонной модели) и потоки данных между ними. В то время как эталонная модель обеспечивает разделение функций, эталонная архитектура отображает эти функции на декомпозицию системы. Соответствие может быть как однозначным, так и неоднозначным. В программном элементе может быть реализована как отдельная часть функции, так и несколько функций сразу.

Эталонные модели, архитектурные образцы и эталонные архитектуры не являются вариантами архитектуры; это не более чем полезные понятия, способствующие фиксации отдельных элементов архитектуры. Каждый из них появляется как результат проектных решений, принимаемых на самых ранних этапах. Отношение между этими проектными элементами представлено на рисунке 4.2.

Рисунок 4.2 - Отношения между эталонными моделями, архитектурными образцами, вариантами эталонной и программной архитектуры

Аналогии архитектуры с другими значениями этого слова проводятся весьма часто. Как правило, архитектура ассоциируется с физической структурой (здания, улицы, аппаратура) и физическим расположением. Архитектор, разрабатывающий план здания, имеет целью обеспечить его доступность, эстетическую привлекательность, освещенность, эксплуатационную надежность и др. Программный архитектор в процессе проектирования системы должен стремиться к обеспечению таких характеристик, как параллелизм, модифицируемость, практичность, безопасность и т. д.; кроме того, он призван установить баланс между требованиями к этим характеристикам.

Аналогии между зданиями и программными системами не очень надежны, они слишком быстро оказываются несостоятельными. Хороши они тем, что помогают осознать важность позиции наблюдателя и прийти к выводу о множественности значений понятия «структура» в зависимости от мотивов ее изучения. Точное определение программной архитектуры значительно менее существенно, чем анализ сущности этого понятия.

4.1.3.2 Архитектурные структуры и представления

Современные программные системы настолько сложны, что разбирать их в комплексе крайне сложно. Приходится концентрировать внимание на одной или нескольких структурах программной системы. Для того чтобы рассуждать об архитектуре осмысленно, мы должны определиться с тем, какая структура или какие структуры в данный момент являются предметом обсуждения, о каком представлении (view) архитектуры мы говорим.

Рассматривая представление архитектуры, мы будем употреблять связанные между собой понятия структуры (structure) и представления (view). Представление – это отображение ряда связанных архитектурных элементов в том виде, в котором ими оперируют заинтересованные в системе лица.

В нем фиксируются отображения совокупности элементов и установленных между ними связей. Структура же – это собственно ряд элементов, существующих в рамках программного или аппаратного обеспечения. В частности, модульная структура представляет собой набор модулей системы с указанием их организации. Модульное представление есть отображение этой структуры, документированное и применяемое теми или иными заинтересованными лицами. Несмотря на то, что эти термины иногда используются как синонимы, мы намерены придерживаться приведенных определений.

Архитектурные структуры подразделяются на три общие группы, в каждую из которых включается элементы определенного характера.

Модульные структуры. Элементами таких структур являются модули – блоки реализации.

Модули предполагают рассмотрение системы с точки зрения кода. Им, как отдельным областям, выделяются определенные функциональные обязанности. Особого внимания тому, как конечное программное обеспечение заявит себя в период прогона, в данном случае не уделяется. Модульные структуры позволяют отвечать на такие вопросы, как: Какие основные функциональные обязанности несет данный модуль? К каким программным элементам он может обращаться? Какое программное обеспечение он фактически использует? Между какими модулями установлены отношения обобщения или специализации (например, наследования)?

Структуры «компонент и соединитель». В данном случае элементами являются компоненты (основные единицы вычислений) и соединители (инструменты взаимодействия между компонентами) периода прогона. Среди вопросов, на которые отвечают структуры «компонент и соединитель», – такие, например, как: Каковы основные исполняемые компоненты и как происходит их взаимодействие? Каковы основные совместно используемые хранилища данных? Какие части системы воспроизводятся? Каким образом по системе проходят данные? Какие элементы системы способны исполняться параллельно? Какие структурные изменения происходят с системой во время ее исполнения?

Структуры распределения. Структуры распределения демонстрируют связь между программными элементами, с одной стороны, и элементами одной или нескольких внешних сред, в которых данное программное обеспечение создается и исполняется, – с другой. Они отвечают на вопросы: на каком процессоре исполняется данный программный элемент? в каких файлах каждый элемент хранится в ходе разработки, тестирования и конструирования системы? каким образом программные элементы распределяются между группами разработчиков?

Программные структуры. Наиболее распространенные и полезные программные структуры представлены на рисунке 4.3.

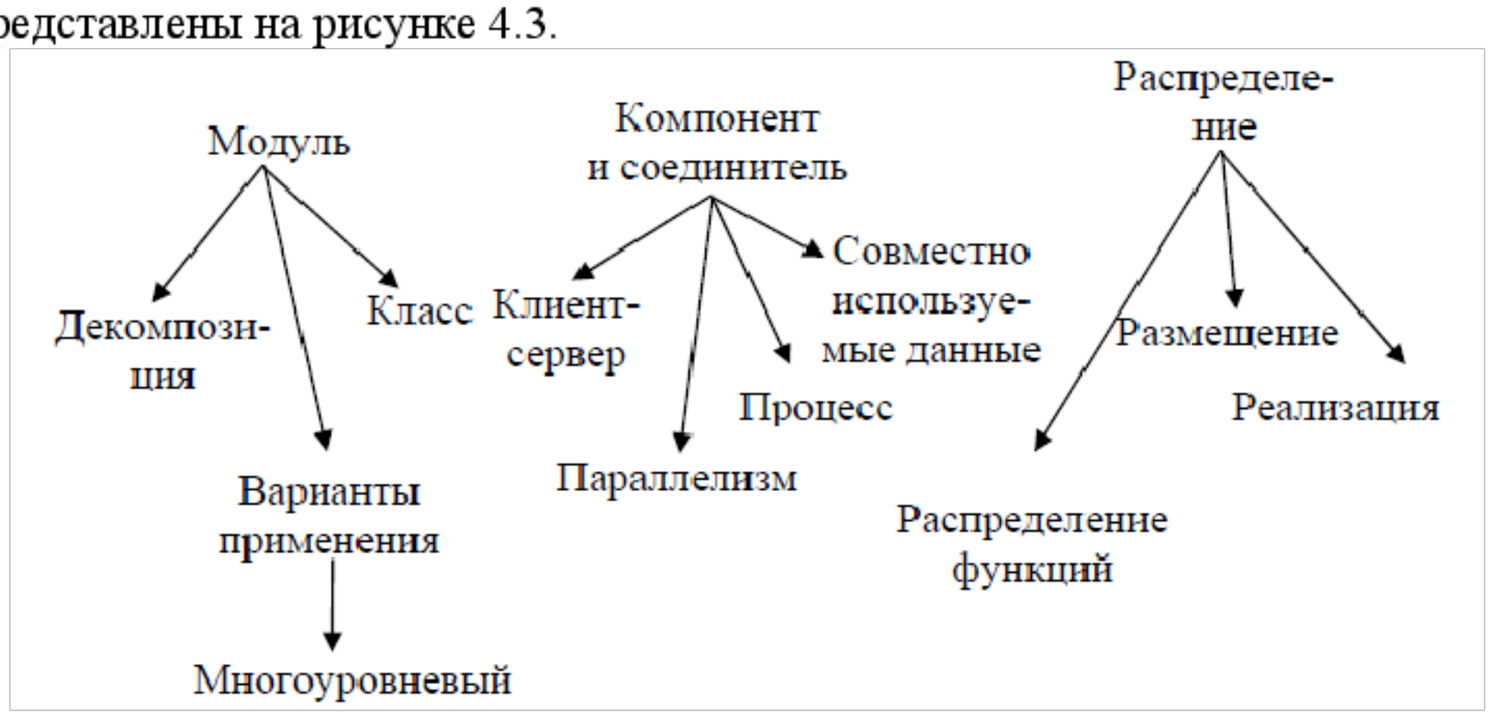


Рисунок 4.3 - Стандартные структуры программной архитектуры

Модуль. Модульные структуры делятся на следующие разновидности.

Декомпозиция. В качестве блоков выступают модули, между которыми установлены отношения «является подмодулем...»; таким образом, крупные модули в рекурсивном порядке разлагаются на меньшие, и процесс этот завершается только тогда, когда мелкие модули становятся вполне понятными. Модули в рамках этой структуры часто используются в качестве отправной точки для последующего проектирования – архитектор перечисляет блоки, с которыми ему предстоит работать, и в расчете на более подробное проектирование, а также, в конечном итоге, на реализацию распределяет их между модулями. У модулей во многих случаях есть связанные продукты (например, спецификации интерфейсов, код, планы тестирования и т. д.). Структура декомпозиции в значительной степени обеспечивает модифицируемость системы – при этом складывается ситуация, когда наиболее вероятные изменения приходятся на долю нескольких небольших модулей. Довольно часто декомпозиция задействуется как основа для организации проекта, включающая структуру документации, планы интеграции и тестирования. Иногда блоки этой структуры получают оригинальные названия (от пользующихся ими организаций). К примеру, в ряде стандартов Министерства обороны США определяются элементы конфигурации компьютерных программ (Computer Software Configuration Items, CSCI) и компоненты компьютерных программ (Computer Software Components, CSC), которые представляют собой не что иное, как блоки модульной декомпозиции.

Варианты использования. Блоками этой важной, но не слишком распространенной структуры могут быть либо модули, либо (в случаях, когда требуется более мелкая структура)

процедуры или ресурсы интерфейсов модулей. Между такими блоками устанавливаются отношения использования (uses relationship). Если для обеспечения правильности первого блока требуется наличие правильной версии (в отличие от заглушки) второго, то последний используется первым. Структура использования полезна при конструировании систем, которые либо легко расширяются дополнительными функциями, либо предполагают возможность быстрого извлечения полезных функциональных подмножеств. Способность без труда разбить рабочую систему на ряд подмножеств подразумевает возможность инкрементной разработки – многофункционального конструкторского приема.

Многоуровневая. Если отношения использования в рамках этой структуры находятся под строгим, особым образом осуществляемым контролем, возникает система уровней – внутренне связанных наборов родственных функций. В рамках строго многоуровневой структуры уровень *n* может обращаться к услугам только в том случае, если они предоставляются уровнем *n*–1. На практике это правило существует в виде многочисленных вариантов (которые частично снимают представленное структурное ограничение). Уровни во многих случаях проектируются в виде абстракций (виртуальных машин), которые, стараясь обеспечить переносимость, скрывают детали реализации нижележащих уровней от вышележащих.

Класс, или обобщение. Блоки модулей в рамках этой структуры называются классами. Отношения между ними строятся по образцам: «наследует от...» и «является экземпляром...». Данное представление способствует анализу коллекций сходного поведения или сходных возможностей (например, классов, которые наследуют от других классов) и параметрических различий, фиксация которых производится путем определения подклассов. Структура классов также позволяет анализировать вопросы повторного использования и инкрементного введения функциональности.

Компонент и соединитель. Среди структур данного вида выделяются следующие:

Процесс или сообщающиеся процессы. Подобно всем прочим структурам «компонент и соединитель», эта является ортогональной по отношению к модульным структурам, а связана она с динамическими аспектами исполняемой системы. В качестве блоков в данном случае выступают процессы или потоки, связь между которыми устанавливается путем передачи данных, синхронизации и/или операций исключения. Здесь, как и во всех остальных структурах «компонент и соединитель», действует отношение прикрепления (attachment), демонстрирующее связь компонентов друг с другом. Структура процессов существенно облегчает конструирование рабочей производительности и готовности системы.

Параллелизм. Данная структура «компонент и соединитель» позволяет архитекторам выявлять перспективы параллелизма и локализовывать возможности состязаний за ресурсы. В качестве блоков выступают компоненты, а соединители играют роль логических потоков. Логическим потоком называется такая последовательность вычислений, которую впоследствии, в ходе процесса проектирования, можно связать с отдельным физическим потоком. Структура параллелизма задействуется на ранних этапах процесса проектирования, способствуя выявлению требований к организации параллельного исполнения.

Совместно используемые данные, или репозиторий. В состав данной структуры входят компоненты и соединители, обеспечивающие создание, хранение и обращение к данным постоянного хранения. Она наилучшим образом приспособлена к таким ситуациям, когда система структурирована на основе одного или нескольких репозиториев совместно используемых данных. Она отражает производство и потребление данных программными элементами периода прогона, а в деле обеспечения высокой производительности и целостности данных эти сведения представляют большую ценность.

Клиент-сервер. Эта структура предназначена для систем, сконструированных в виде группы взаимодействующих клиентов и серверов. В качестве компонентов в данном случае выступают клиенты и серверы, а соединителями являются протоколы и сообщения, которыми они обмениваются в процессе обеспечения работоспособности системы. Такая структура требуется для разделения задач (обеспечивающего модифицируемость), физического распределения и выравнивания нагрузок (обеспечивающего эффективность исполнения).

Распределение. Среди структур распределения выделяются следующие:

Размещение. Структура размещения отражает распределение программного обеспечения между элементами аппаратной обработки и передачи данных. В качестве распределяемых элементов могут выступать программные продукты (как правило, процессы из представления «компонент и соединитель»), аппаратные объекты (процессоры) и каналы передачи

данных. Отношения устанавливаются по распределению и демонстрируют физические устройства, на которых размещаются программные элементы; возможны также отношения миграции, однако они устанавливаются только в случае динамического распределения. Настоящее представление позволяет инженерам анализировать производительность, целостность данных, готовность и безопасность. Все эти характеристики чрезвычайно важны в условиях распределенных и параллельных систем.

Реализация. Данная структура демонстрирует отображение программных элементов (обычно модулей) на файловую структуру (структуры) в условиях разработки системы, интеграции и управления конфигурациями. Это крайне важно в контексте организации разработки и процессов конструирования.

Распределение функций. Данная структура обеспечивает разделение обязанностей по реализации и интеграции модулей между соответствующими группами разработчиков. Наличие в составе архитектуры структуры распределения функций делает очевидным, что при принятии соответствующих решений учитывались как архитектурные, так и организационные факторы. Архитектору должно быть известно, какие навыки требуются от разных групп разработчиков. Кроме того, если речь идет о масштабных распределенных проектах с несколькими источниками, на основе структуры распре- деления функций, можно выявлять функциональные сходства и назначать их одной группе разработчиков, отказываясь, таким образом, от их стихийной многократной реализации.

Общая схема программных структур приводится в таблице 4.1. В ней рассматривается значения элементов, отношения, характерные для каждой из структур, и варианты их практического применения.

Таблица 4.1 - Архитектурные структуры системы

Программная структура	Отношения	Варианты практического применения
Декомпозиция	«Является подмодулем...»; «пользуется скрытой информацией совместно с...»	Распределение ресурсов, структурирование и планирование проекта; информационная закрытость, инкапсуляция; управление конфигурациями
Варианты использования	«Требует наличия...»	Конструирование подмножеств; инженерные расширения
Многоуровневая	«Требует наличия...», «обращается к услугам...», «обобщает...»	Инкрементная разработка; реализация систем на основе переносимости «виртуальных машин»
Классы	«Является экземпляром...», «использует метод доступа из...»	В объектно-ориентированных системах проектирования, производящих на основе универсального шаблона быстрые, почти идентичные реализации
Клиент-сервер	«Обменивается данными с...», «зависит от...»	Распределенное функционирование; разделение задач; анализ производительности; выравнивание нагрузки
Процесс	«Исполняется параллельно с...», «может исполняться параллельно с...», «исключает», «предшествует» и т. д.	Анализ сроков; анализ производительности
Параллелизм	«Исполняется в одном логическом потоке»	Выявление местоположений в которых потоки могут разветвляться, объединяться, создаваться и уничтожаться
Совместно используемые данные	«Производит данные», «потребляет данные»	Производительность; целостность данных, модифицируемость
Размещение	Распределение, миграция	Анализ производительности, готовности и защиты
Реализация	«Хранится в...»	Управление конфигурациями, интеграция, тестирование
Распределение функций	«Назначается...»	Управление проектом, оптимальное использование интеллектуальных ресурсов, управление общностью

Как правило, структура системы анализируется с точки зрения ее функциональности; при этом мы забываем о других ее свойствах: физическом распределении, взаимодействии процессов и синхронизации – все эти свойства обязательно учитываются на уровне архитектуры. Каждая структура содержит метод анализа тех или иных значимых атрибутов качества. К примеру, для того чтобы создать легко расширяемую или сокращаемую систему, необходимо *сконструировать* (именно сконструировать, а не просто зафиксировать) структуру использования. Структура процессов *конструируется* с целью исключения

взаимоблокировки и расширения узких мест. Структура декомпозиции модулей *конструируется* в расчете на производство модифицируемых систем и т. д. Каждая подобная структура снабжает архитектора оригинальным представлением системы и дополнительной базовой точкой проектирования.

4.2 Проектирование архитектуры

4.2.1 Атрибутный метод проектирования

Атрибутный метод проектирования (Attribute-Driven Design, ADD) – это методика определения программной архитектуры, в которой процесс декомпозиции основывается на предполагаемых атрибутах качества продукта. Это рекурсивный процесс декомпозиции, на каждом из этапов которого происходит отбор тактик и архитектурных образцов, удовлетворяющих тем или иным сценариям качества, а также распределение функциональности, направленное на конкретизацию типов модулей данного образца. В контексте жизненного цикла ADD располагается сразу после анализа требований, а начинается он, как мы уже говорили, в тот момент, когда об архитектурных мотивах можно говорить с достаточной степенью уверенности.

Результатом применения ADD являются первые несколько уровней представления декомпозиции модулей архитектуры, а также все другие связанные с ними представления. Не стоит, впрочем, думать, что после ADD становятся известны все детали представлений, – система описывается как набор контейнеров функциональности и существующих между ними взаимосвязей. Во всем процессе проектирования это первый случай сочленения архитектуры, результаты которого, естественно, весьма приблизительны. Тем не менее, в контексте реализации предполагаемых атрибутов качества это очень важный момент, поскольку именно здесь закладываются основы достижения функциональности. Различие между архитектурой, являющейся результатом выполнения ADD, и архитектурой, готовой к реализации, лежит в плоскости принятия подробных проектных решений. В частности, это может быть выбор между конкретными объектно-ориентированными образцами проектирования, с одной стороны, и промежуточным программным обеспечением, применение которого сопряжено с многочисленными архитектурными ограничениями, – с другой. Архитектура, спроектированная средствами ADD, предусматривает отсрочку принятия этого решения, благодаря чему достигается большая гибкость.

С участием общих сценариев, а также тактик и образцов можно создать ряд различных процессов проектирования. Отличаются они принятыми принципами деления проектных работ и содержанием процесса проектирования.

4.2.1.1 Этапы ADD

1. *Выбор модуля для декомпозиции*. Как правило, в качестве исходного модуля берется система в целом. Все необходимые входные данные (ограничения, функциональные требования и требования к качеству) для него должны быть известны.

2. Уточнение модуля в несколько этапов:

а) выбор архитектурных мотивов из набора конкретных сценариев реализации качества и функциональных требований. На этом этапе определяются наиболее важные в контексте проведения декомпозиции моменты;

б) выбор архитектурного образца, соответствующего архитектурным мотивам. Создание (или выбор) образца, тактики которого позволяют реализовать эти мотивы. Выявление дочерних модулей, необходимых для реализации этих тактик;

в) конкретизация модулей, распределение функциональности из элементов Use Case, составление нескольких представлений;

г) определение интерфейсов дочерних модулей. Декомпозиция имеет своим результатом новые модули, а также ограничения на типы взаимодействия между ними. Для каждого из модулей эти сведения следует зафиксировать в документации по интерфейсу;

д) проверка и уточнение элементов Use Case в сценариях качества и наложение, исходя из них, ограничений на дочерние узлы. На этом этапе мы проверяем, все ли факторы учли, а также, в расчете на последующую декомпозицию или реализацию, подготавливаем дочерние модули.

3. Вышеперечисленные этапы следует выполнить в отношении всех нуждающихся в дальнейшей декомпозиции модулей.

4.2.2 Создание макета системы

После проектирования архитектуры и формирования рабочих групп можно приступать к конструированию макета системы. Цель этого этапа в том, чтобы обеспечить основополагающую возможность реализации функциональности системы в предпочтительном (с точки зрения задач проекта) порядке.

Согласно классической методике программной инженерии, следует «заглушать» секции кода так, чтобы различные части системы можно было вводить в действие и тестировать по отдельности. О каких именно частях идет речь? Последовательность реализации всегда можно установить по характеристикам архитектуры.

В первую очередь следует реализовать те программы, которые связаны с исполнением и взаимодействием между архитектурными компонентами.

В системе реального времени это может быть планировщик, в системе на основе правил – процессор правил (с прототипом набора правил), управляющий их активизацией, в многопроцессной системе – механизмы синхронизации процессов, а в клиент-серверной системе – механизм координации действий клиента и сервера. Базовый механизм взаимодействия во многих случаях выражается в виде промежуточных программ стороннего производства; если это так, реализация заменяется установкой. Помимо базовой инфраструктуры передачи данных или взаимодействия имеет смысл ввести в действие ряд простейших функций, инициирующих механическое поведение. На этом этапе в вашем распоряжении уже появляется исполняемая система. Это та основа, на которую можно начинать насаживать дополнительную функциональность.

Теперь выбирайте – какие функциональные элементы следует ввести в эту систему. Решение в данном случае принимается под влиянием нескольких факторов: во-первых, из побуждения снизить риск, обратившись к наиболее трудным областям в первую очередь, во-вторых, исходя из возможностей и специализации имеющегося персонала, и, в-третьих, из соображений «выбросить» продукт на рынок как можно быстрее.

Приняв решение относительно введения в систему очередной порции функциональности, обратитесь к структуре использования – она сообщит о том, какие еще программные средства системы должны корректно исполняться (другими словами, развиться из состояния банальной заглушки в полноценные элементы), для того чтобы реализация выбранных функций стала возможной.

Таким вот образом в систему можно вводить все новые и новые элементы, пока они не закончатся. На любом из таких этапов действия по интеграции и тестированию не представляют большой сложности; равным образом легко обнаружить источник недавно появившихся неисправностей. Чем меньше приращение, тем более предсказуемы бюджет и график и тем больше диапазон решений по поставке у управленцев и специалистов по маркетингу.

Заглушенные элементы, несмотря ни на что, приближают момент завершения работы над системой. Поскольку заглушки относятся к тем же интерфейсам, которые будут присутствовать в окончательной версии системы, даже в отсутствии полноценной функциональности они помогают уяснить и протестировать взаимодействие между компонентами. Проверить это взаимодействие с помощью компонентов-заглушек можно двумя способами: путем генерирования жестко закодированных искусственных выходных данных или считывания таких данных из файла. Кроме того, они способны генерировать искусственную нагрузку, по результатам которой можно приблизительно определить, сколько времени уйдет на фактическую обработку данных в законченной рабочей версии системы. Это помогает на ранней стадии проектирования выявить требования по производительности системы, включая рабочие взаимодействия и узкие места.

Согласно Кусумано (Cusumano) и Селби (Selby), компания Microsoft выстраивает свою стратегию на основе эволюционного жизненного цикла поставки (Evolutionary Delivery Life Cycle). По той версии методики, которой пользуется Microsoft, «завершенный» макет системы создается на раннем этапе жизненного цикла продукта, а впоследствии с некоторой регулярностью (во многих случаях ежедневно), строятся «рабочие» версии с сокращенной функциональностью. В результате получается рабочая система, о достаточности характеристик

которой можно принять решение в любой момент и которую в любой же момент можно развернуть. Есть, впрочем, одна проблема, которую следует предусмотреть, – дело в том, что та рабочая группа, которая заканчивает свою часть системы первой, задает интерфейс, которому должны соответствовать все последующие системы. Это обстоятельство ставит в невыгодное положение наиболее сложные части системы – их разработка сопряжена со значительно большими объемами аналитических действий, вследствие чего вероятность первоочередного определения их интерфейсов сильно снижается. Таким образом, и без этого сложные подсистемы становятся еще более сложными. По нашему убеждению, все согласования по поводу интерфейсов следует проводить на этапе создания макета системы – при таком условии на последующих стадиях разработки можно будет обратить большее внимание на достижение эффективности.

4.3 Документирование программной архитектуры

4.3.1 Варианты применения архитектурной документации

Характер архитектуры любой системы обусловливается предъявляемыми к ней требованиями; это утверждение справедливо и по отношению к документации архитектуры, другими словами, содержание документации зависит от предполагаемых вариантов ее применения. Документация ни при каких обстоятельствах не может быть универсальной. С одной стороны, она должна быть абстрактной и, следовательно, доступной для понимания новыми сотрудниками, но с другой – весьма детальной – настолько, чтобы ее можно было использовать как план проведения анализа. Между архитектурной документацией, предназначенной, скажем, для проведения анализа безопасности, и архитектурной документацией для изучения конструкторами должны существовать серьезные различия. С другой стороны, у этих вариантов будет мало общего с содержанием руководства для ознакомления, предназначенного новому сотруднику.

Архитектурная документация одновременно инструктивна и описательна. Ограничивая диапазон возможных решений, с одной стороны, и перечисляя принятые ранее проектные решения по системе, – с другой, она обязывает соблюдать определенные принципы.

Из всего этого следует, что заинтересованные в составлении документации лица преследуют разные цели – от представленной в ней информации они требуют разной направленности, разных уровней детализации и разных трактовок. Предполагать, что один и тот же архитектурный документ будет одинаково воспринят всеми без исключения заказчиками, наивно. Стремиться следует к тому, чтобы любое заинтересованное лицо смогло быстро найти необходимую информацию, как можно меньше в процессе ее поиска натыкаясь на незначительные (с его точки зрения) сведения.

В некоторых случаях простейшим решением представляется создание нескольких документов, предназначенных для разных заинтересованных лиц. Впрочем, как правило, задача ограничивается созданием единого комплекта документации с дополнительными облегчающими поиск сведений инструкциями.

Согласно одному из фундаментальных правил технической документации в общем и документации программной архитектуры в частности, составитель должен принимать позицию читателя. Без труда составленная, но трудная для восприятия (с точки зрения читателя – в данном случае заинтересованного лица) документация не найдет себе применения.

4.3.2 Представления

Одним из наиболее важных понятий документирования программной архитектуры является «представление» (view). С понятием представления, которое можно кратко определить как способ фиксации структуры, связан основной принцип документирования программной архитектуры.

Документирование архитектуры подразумевает документирование всех значимых представлений с последующей фиксацией сведений, относящихся одновременно к нескольким представлениям.

Принцип этот полезен тем, что он помогает разбить проблему документирования архитектуры на ряд менее обширных элементов:

- выбор значимых представлений;

- документирование представления;
- документирование сведений, относящихся к нескольким представлениям.

4.3.2.1 Выбор значимых представлений

Какие представления следует считать значимыми? Для ответа на этот вопрос необходимо знать заинтересованных лиц и предпочтительные для них способы пользования документацией. Лишь располагая этими сведениями, вы сможете составить удобный для них пакет документации. Любой вариант применения архитектуры как средства постановки задач конструкторов, основы для изучения системы, восстановления ее свойств или планирования проекта можно свести к отдельному заинтересованному лицу, предполагающему задействовать документацию архитектуры соответствующим образом. Не менее серьезное влияние на выбор представлений для дальнейшего документирования оказывают наиценнейшие для большинства заинтересованных в разработке системы лиц атрибуты качества. К примеру, многоуровневое представление (layered view) сообщает сведения о переносимости системы. По представлению размещения (deployment view) можно судить о производительности и надежности системы. И так далее. За отражение этих атрибутов качества в документации ратуют аналитики (а быть может, и сам архи- тектор), в задачу которых входит проверка архитектуры на предмет ее соответствия атрибутам качества.

Различные представления соответствуют отдельным задачам и вариантам использования. Именно по этой причине мы призываем к выбору того или иного представления или набора представлений. Их выбор зависит от задач потребителей документации. Представления ставят акценты на разные элементы системы и/или установленные между ними связи.

Представление отражает произвольный набор элементов системы и установленных между ними отношений. Следовательно, представлением может быть любое сочетание элементов и отношений, которое, по вашему мнению, имеет шансы оказаться интересным части заинтересованных лиц. Ниже приводится состоящая из трех этапов процедура подбора представлений для конкретного проекта.

1. Составьте список возможных представлений. Начните с составления для своего проекта таблицы заинтересованных лиц/представлений. В столбцах выразите применимые к вашей системе представления. Некоторые из них (например, модульное представление и представление вариантов использования) носят универсальный характер; иные (многоуровневое представление, а также большинство представлений из группы «компонент и соединитель», в частности клиент-серверное представление и представление совместно используемых данных) подходят только для соответствующим образом спроектированных систем. Разобравшись с содержанием строк и столбцов, заполните все ячейки, отразив в них степень детализации сведений о каждом представлении, необходимую тем или иным заинтересованным лицам: здесь возможны произвольная детализация, обзор, средняя или высокая детализация.

2. Комбинируйте представления. Скорее всего, количество представлений, отраженных в составленном по результатам первого этапа списке, окажется недопустимым. Для того чтобы сократить список, найдите в таблице представления, требующие не более чем обзорной детализации или служащие ограниченному числу заинтересованных лиц. Проверьте, нельзя ли удовлетворить их запросы другим, более универсальным представлением. Затем найдите представления, которые можно комбинировать — выразить в одном сводном представлении информацию из нескольких исходных представлений. В небольших по масштабу проектах информативное содержание представления реализации, как правило, адекватно отражается в представлении декомпозиции на модули. Последнее, в свою очередь, хорошо сочетается с представлениями вариантов использования и многоуровневым представлением. Наконец, представление размещения можно скомбинировать с любым представителем группы «компонент и соединитель», отражающим распределение компонентов между аппаратными элементами, например с представлением процессов.

3. Расставляйте приоритеты. Представления, оставшиеся после выполнения второго этапа, должны соответствовать потребностям сообщества заинтересованных лиц. Теперь необходимо решить, какие из этих представлений имеют первостепенное значение. Конкретное решение зависит от деталей проекта; впрочем, вы в любом случае должны помнить, что полностью завершать работу над одним представлением, прежде чем приступать к следующему, совершенно не обязательно. Информация, детализированная на уровне обзора, представляет

некоторую ценность, так что наши предпочтения на стороне метода разработки материала «в ширину». Кроме того, интересы одних заинтересованных лиц в ряде случаев ставятся выше интересов других.

4.3.3 Документирование представления

Стандартного шаблона документирования представлений не существу- ет. Поэтому ниже речь пойдет о методике, доказавшей свою жизнеспособ- ность в практической деятельности – стандартной семичастной структуре (seven-part standard organization). Во-первых, вне зависимости от того, каким разделам вы отдадите предпочтение, ввести стандартную структуру совершенно необходимо. Распределение информации по отдельным разделам помогает составителю документации уверенно приступить к решению задачи и установить момент ее выполнения; что же касается читателя, то ему становится проще быстро отыскать интересующую информацию и пропустить все ненужное.

1. Первичное отображение (primary presentation) содержит перечень элементов представления и установленные между ними отношения. В первичном представлении должна содержаться (в форме словаря) та информация о системе, которую необходимо донести до читателя в первую очередь. Это, без сомнения, основные элементы и отношения представления, хотя в некоторых случаях они могут быть отражены не полностью. К примеру, в первичном отображении можно показать элементы и отношения, характерные для нормального режима работы, а сведения об обработке ошибок или исключений представить во вспомогательной документации.

Как правило, первичное отображение составляется в графической форме. Дело в том, что большинство графических нотаций годятся только для первичного отображения – для всех остальных элементов документации они не приспособлены. Неизменным спутником графического представления должен быть перечень условных обозначений, содержащий объяснение использованной нотации и символики или хотя бы указывающий на источники получения этих объяснений.

Иногда первичное отображение составляется в виде таблицы, которая по своему характеру прекрасно подходит для компактного представления больших объемов информации. Требование о кратком выражении наиболее значимых сведений о представлении распространяется и на текст.

2. Каталог элементов (element catalog) предназначен для более детального описания элементов и отношений как участвующих, так и не участвующих в первичном отображении. Архитекторы часто совершают одну и ту же ошибку – уделяя излишне серьезное внимание составлению первичного отображения, они забывают, что без дополнительной информации в нем мало толку. К примеру, если на диаграмме показаны элементы A, B и C, необходимо составить относительно детальное описание сущности этих элементов, их назначения и ролей, причем разместить эту информацию лучше всего в словаре представления. Скажем, для представления декомпозиции на модули характерно наличие элементов-модулей, различных вариантов отношения «является частью», а также свойств, определяющих обязанности каждого модуля. В представлении процессов содержатся элементы-процессы, отношения, определяющие синхронизацию и другие механизмы межпроцессного взаимодействия, а также свойства с временными параметрами.

Кроме того, в каталоге обязательно должны разъясняться все элементы и отношения, значимые для данного представления, но по каким-то причинам не показанные в первичном отображении.

3. На контекстной диаграмме (context diagram) должно быть показано отношение изображенной в представлении системы к своему окружению из словаря представления. К примеру, представление «компонент и соединитель» подразумевает демонстрацию взаимодействия отдельных компонентов и соединителей с внешними компонентами и соединителями через посредство интерфейсов и протоколов.

4. Руководство по изменчивости (variability guide) демонстрирует способы применения изменяемых параметров, входящих в состав показанной в данном представлении архитектуры. В некоторых архитектурах принятие решений откладывается до более поздних стадий процесса разработки, что не отменяет необходимость в составлении документации. Примеры изменчивости обнаруживаются во всех линейках программных продуктов, архитектура которых

предусматривает возможность создания множества конкретных систем. В руководстве по изменчивости следует документировать все изменяемые параметры архитектуры, включая: альтернативы, рассматриваемые при принятии того или иного решения.

В модульном представлении такими альтернативами являются различные варианты параметризации модулей. В представлении «компонент и соединитель» могут быть отражены ограничения по дублированию, планированию или выбору протоколов. В представлении распределения это условия назначения конкретного программного элемента тому или иному процессору; время связывания альтернатив. Одни решения принимаются в период проектирования, другие – в период производства, третьи – в период исполнения.

5. Предпосылки архитектурного решения (architecture background) – это раздел, в котором содержится обоснование отраженного в данном представлении проектного решения. Цель его – объяснить читателю, почему проект выглядит так, как он выглядит, и представить убедительные аргументы его превосходства. В состав этого раздела входят следующие элементы:

- логическое обоснование, демонстрирующее предпосылки принятия проектных решений, отраженных в данном представлении, и причины отказа от альтернатив;
- результаты анализа, оправдывающие проектное решение и объясняющие последствия модификаций;
- отраженные в проектном решении допущения.

6. Глоссарий терминов (glossary of terms), применяемых в представлениях, и краткое описание каждого из них.

7. Другая информация. Содержание этого раздела зависит от методов работы конкретной организации. В частности, здесь можно разместить административные сведения: авторство, данные управления конфигурациями и историю изменений. Кроме того, архитектор волен разместить здесь систему ссылок на отдельные разделы сводки требований. Таким образом, речь идет об информации, которая, строго говоря, не имеет прямого отношения к архитектуре, но которую удобнее всего привести вместе с архитектурными сведениями.

4.3.3.1 Документирование поведения

Представления содержат структурную информацию о системе. Но для рассуждений о некоторых свойствах системы ее недостаточно. К примеру, для того чтобы уяснить вопросы взаимоблокировки, необходимо знать последовательность взаимодействий между элементами, которую структурная информация не отражает. Эти сведения, равно как возможности параллелизма и временные зависимости, характерные для взаимодействий (которые происходят в установленные моменты или по истечении установленных временных периодов), раскрывают поведенческие описания. Поведение документируется применительно к отдельному элементу или к множеству взаи- модействующих элементов. Выбор в вопросах моделирования зависит от типа проектируемой системы. К примеру, если речь идет о встроенной системе реального времени, на первое место выходят свойства времени и время наступления событий. В системе банковского обслуживания значительно важнее фактического времени наступления событий оказывается их последовательность (например, последовательность элементарных транзакций и процедур отката). В зависимости от вида предполагаемых аналитических действий уместно обращаться к разным методикам моделирования и нотациям. Примерами поведенческих описаний в языке UML являются диаграммы последовательностей и схемы состояний. Подобные нотации весьма широко распространены.

Схемы состояний как формализм появились в 1980-х гг. и изначально предназначались для описания реактивных систем. Ряд реализованных в них полезных расширений дополняет традиционные диаграммы состояний (такие, как вложенность состояний и состояния «и») и придает выразительность абстракции и параллелизму моделей. Схемы состояний позволяют рассуждать о системе в целом. Предполагается отображение всех ее состояний, а методики анализа в отношении системы приобретают универсальный характер. Становятся возможными ответы на вопрос типа: всегда ли время отклика на данный стимул будет меньше 0,5 секунды?

Диаграмма последовательностей помогает документировать последовательность обменов стимулами. Она отражает кооперацию применительно к экземплярам компонентов и их взаимодействию, причем последнее представляется во временной последовательности. Вертикальное измерение при этом выражает время, а горизонтальное – различные компоненты. Диаграммы последовательностей позволяют строить умозаключения на основе конкретных

сценариев использования. Они демонстрируют механизм реагирования системы на отдельные стимулы, иллюстрируют выбор путей в системе и отвечают на вопрос типа: какие параллельные операции приходят в момент реагирования системы на определенные стимулы в определенных условиях?

4.3.3.2 Документирование интерфейсов

Интерфейс (interface) – это граница, на которой встречаются, взаимодействуют или передают друг другу информацию две независимые сущности.

Очевидно, что интерфейсы элементов – носителей их внешне видимых другим элементам свойств – являются понятием архитектурным. Поскольку без них невозможны ни анализ, ни проектирование систем, документировать интерфейсы совершенно необходимо.

Под документированием интерфейса подразумевается указание его имени, идентификация, а также отражение всех синтаксических и семантических сведений о нем. Первые два элемента – указание имени и идентификация – обобщенно называются «сигнатура» интерфейса. Если в качестве ресурсов интерфейса выступают вызываемые программы, сигнатура обозначает их и определяет их параметры. Параметры определяются по порядку, типу данных и (иногда) по принципу возможности изменения их значений программой. Та информация о программе, которая содержится в сигнатуре, обычно приводится в заголовочных файлах C или C++ и в интерфейсах Java.

При всей полезности сигнатур (в частности, они делают возможной автоматическую проверку конструкций) ими все не исчерпывается. Соответствие сигнатур обеспечивает успешную компиляцию и/или компоновку системы, но совершенно не гарантирует достижение конечной цели – ее нормальное функционирование. Необходимая для этого информация относится к семантике интерфейса, которая сообщает, что происходит при активизации ресурсов.

Интерфейс документируется в форме спецификации – изложения свойств элемента, которые архитектор желает предать огласке. Это должна быть только та информация, которая необходима для организации взаимодействия с интерфейсом. Другими словами, архитектор должен решить, во-первых, какую информацию об элементе допустимо и уместно сообщить читателю и, во-вторых, какая информация, вероятнее всего, не будет подвержена изменениям. В процессе документирования интерфейса важно, с одной стороны, не раскрыть слишком много сведений, и с другой – не утаить необходимые данные. Разработчики не смогут наладить успешное взаимодействие с элементом, если о нем будет недостаточно информации. Избыток информации, в свою очередь, усложняет будущие изменения в системе и усиливает их влияние, делает интерфейс неудобным для восприятия. Для того чтобы достойно справиться с этой ситуацией, наибольшее внимание следует уделять не реализации элементов, а их взаимодействию с рабочими средами. Документированию подлежат только внешне видимые явления.

Элементы, присутствующие в виде модулей, часто напрямую соответствуют одному или нескольким элементам представления «компонент и соединитель». Как правило, интерфейсы элементов в представлении модулей и представлении «компонент и соединитель» схожи или идентичны и документировать их в обоих этих представлениях излишне. Поэтому спецификацию интерфейса следует привести в модульном представлении, а в «компоненте и соединителе» поместить ссылку на нее и изложить индивидуальную для данного представления информацию. Кроме того, один модуль может оказаться общим для нескольких модульных представлений – например, декомпозиции на модули и использования. В таком случае, как и в предыдущем, спецификацию интерфейса следует привести в одном из этих представлений, а во всех остальных поставить соответствующую ссылку.

Шаблон для документирования интерфейсов. Ниже изложен один из вариантов стандартной структуры документации интерфейса. Некоторые ее элементы, если они оказываются бесполезными в конкретном применении, можно выкинуть, другие, наоборот, ввести. Значительно более важными, чем сама стандартная структура, представляются нам способы ее применения. Ваша цель должна заключаться в том, чтобы в точности изложить все внешне видимые взаимодействия интерфейсов, предусмотренных в проекте.

Индивидуальность интерфейса. Если у элемента несколько интерфейсов, их нужно как-то отличать друг от друга. Как правило, для этого интерфейсам присваиваются имена, в некоторых случаях сопровождающиеся указанием номера версии.

Предоставляемые ресурсы. Основным предметом документирования любого интерфейса должны быть предоставляемые им ресурсы. Они определяются синтаксисом, семантикой (описывающей следствия их применения) и ограничениями по использованию. Существует ряд нотаций, предназначенных для документирования синтаксиса интерфейса. Одна из них – язык описания интерфейсов (interface definition language, IDL), разработанный рабочей группой OMG, – применяется в сообществе CORBA. С помощью специальных языковых конструкций этого языка описываются типы данных, операции, атрибуты и исключения. Семантическую информацию можно выразить только в комментариях. В большинстве программных языков есть встроенные средства спецификации сигнатур элементов, примером чему – заголовочные файлы (.h) в C и спецификации пакетов в Ada. Наконец, на языке UML синтаксическая информация об интерфейсе выражается через стереотип interface. Для интерфейса необходимо хотя бы ввести имя; кроме того, архитектор волен составить для него сигнатуру.

Синтаксис ресурса. Здесь указывается сигнатура ресурса. Содержащейся в сигнатуре информации должно быть достаточно для того, чтобы любая сторонняя программа смогла корректно с точки зрения синтаксиса обратиться к программе, использующей данный ресурс. Таким образом, в сигнатуру входят имя ресурса, имена и логические типы данных его аргументов (если таковые имеются) и т. д.

Семантика ресурса. Здесь приводится описание результатов вызова ресурса, в частности:

- присваивание значений данным, к которым может обращаться актер, вызывающий рассматриваемый ресурс – от простого задания значения возвращаемого аргумента до обновления центральной базы данных;
- события и сообщения, сигнализируемые/отправляемые в результате обращения к ресурсу;
- предполагаемое поведение других ресурсов как результат обращения к рассматриваемому ресурсу (к примеру, если данный ресурс уничтожит некий объект, то последующие попытки обращения к этому объекту будут приводить к новому результату – ошибке);
- результаты, наблюдаемые пользователем (встречающиеся в основном во встроенных системах: скажем, вызов программы, ответственной за включение бортового индикатора, приводит к наблюдаемому результату).

Кроме того, семантика должна прояснять вопросы, связанные с исполнением ресурса: будет ли он носить элементарный характер, можно ли его приостановить или прервать. Как правило, семантическая информация выражается средствами естественного языка. Для фиксации предусловий и постусловий – сравнительно простого и эффективного метода выражения семантики – специалисты во многих случаях прибегают к булевой алгебре. Еще одним средством передачи семантической информации являются следы – запи- си последовательностей операций и взаимодействий, описывающих реакцию элемента на тот или иной вариант использования.

Ограничения на использование ресурсов. В каких обстоятельствах возможно обращение к рассматриваемому ресурсу? Существует ли необходимость в предваряющей его считывание инициализации данных? Обусловливается ли вызов одного метода вызовом другого? Не установлены ли какието ограничения относительно количества актеров, имеющих возможность одномоментного взаимодействия с ресурсом? Возможно, в силу принадлежности элемента определенному актеру, он единственный, кто обладает пол- номочиями по модификации элемента, а все остальные актеры ограничиваются лишь считыванием. Не исключено также, что, согласно многоуровневой схеме безопасности, каждый актер может обращаться к строго определенным ресурсам или интерфейсам. Если рассматриваемый ресурс отталкивается от каких-либо допущений в своем окружении (например, требует наличия других ресурсов), их следует задокументировать.

Определения типов данных. Если какой-либо ресурс интерфейса задействует тип данных, отличный от предусмотренного соответствующим языком программирования, архитектор должен привести определение типа данных. Если он определен в другом элементе, достаточно установить ссылку на его документацию. Как бы то ни было, программисты, которые пишут элементы, обращаясь к такому ресурсу, должны знать:

а) как объявлять переменные и константы типа данных;

б) как в рамках этого типа данных записывать литеральные значения;

в) какие операции и сравнения применимы к членам типа данных;

г) как преобразовывать значения этого типа данных в данные других типов.

Определения исключений. Здесь предполагаются описания исключений, которые могут порождать ресурсы на данном интерфейсе. Поскольку одни и те же исключения во многих случаях характерны для множества ресурсов, имеет смысл отделять список исключений каждого ресурса от их определений, причем последние размещать в специальном словаре. Настоящий раздел как раз исполняет роль словаря. Здесь же допустимо определять стандартное поведение обработки ошибок.

Характерная для интерфейса изменчивость. Предполагает ли данный интерфейс возможность конфигурирования элемента? Подобные параметры конфигурации (configuration parameters), а также их воздействие на семантику интерфейса подлежат документированию. В качестве примеров изменчивости можно привести емкость видимых структур данных и рабочие характеристики соответствующих алгоритмов. Для каждого параметра конфигурации архитектор должен зафиксировать имя, диапазон значений и время связывания его фактических значений.

Характеристики атрибутов качества интерфейса. Архитектор должен задокументировать характеристики атрибутов качества (например, производительность или надежность), предоставляемых интерфейсом конечным пользователям. Эти сведения можно выразить в форме ограничений на реализацию составляющих интерфейс элементов. Выбор атрибутов качества, на которых предполагается сконцентрироваться и принять обязательства, проводится в зависимости от контекста.

Требования к элементам. Среди требований элемента может значиться наличие конкретных именованных ресурсов других элементов. Документация в данном случае составляется так же, как и в отношении ресурсов, – указываются синтаксис, семантика и ограничения по использованию. В некоторых случаях подобного рода информацию удобнее документировать в форме ряда допущений о системе, принятых проектировщиком элемента.

В таком случае требования можно представить на суд экспертов, которые уже на ранних стадиях процесса проектирования смогут подтвердить или опровергнуть принятые допущения.

Логическое обоснование и соображения о проекте. Как и в случае с логическим обоснованием архитектуры в целом (или архитектурных представлений), архитектор должен документировать факторы, обусловившие принятие тех или иных проектных решений относительно интерфейса. В логическом обосновании необходимо указать мотивацию принятия этих решений, ограничения и компромиссы, обрисовать рассмотренные, но впоследствии забракованные решения (не забыв попутно изложить причины отказа от них), и изложить соображения архитектора касательно дальнейших изменений интерфейса.

Руководство по использованию. В некоторых случаях требуется анализ семантики с позиции связей между множеством отдельных взаимодействий.

Если это так, в дело вступает протокол (protocol), документировать который следует согласно последовательности взаимодействий. Протоколы отражают комплексное поведение при взаимодействии тех образцов использования, которые, по мысли архитектора, должны встречаться с определенной регулярностью. Сложность взаимодействия с элементом через его интерфейс следует компенсировать статической поведенческой моделью (например, схемой состояний) или примерами проведения отдельных видов взаимодействия (в форме диаграмм последовательностей).

4.4 Методы анализа архитектуры

4.4.1 Метод анализа компромиссных архитектурных решений – комплексный подход к оценке архитектуры

Метод анализа компромиссных архитектурных решений (Architecture Tradeoff Analysis Method, ATAM) – комплексная универсальная методика оценки программной архитектуры. В соответствии с названием этот метод обнаруживает степень реализации в архитектуре тех или иных задач по качеству, а также (исходя из допущения о том, что любое архитектурное решение влияет сразу на несколько задач по качеству) механизм их взаимодействия – другими словами, их взаимозаменяемость.

Основное назначение ATAM состоит в том, чтобы выявить коммерческие задачи, поставленные в контексте разработки системы и проектирования архитектуры. Вкупе с

участием заинтересованных лиц это помогает специалистам по оценке сфокусироваться на тех элементах архитектуры, которые играют первостепенную роль для реализации упомянутых задач.

4.4.1.1 Этапы ATAM

Операции оценки по методу ATAM распадаются на четыре этапа.

На нулевом этапе – «Установление партнерских отношений и подготовка» – руководители группы оценки проводят неофициальные совещания с основными ответственными за проект лицами и прорабатывают подробности предстоящей работы. Представители проекта посвящают специалистов по оценке в суть проекта, тем самым повышая квалификацию некоторых из них. Эти две группы принимают согласованные логистические решения: где и когда встречаться, кто предоставит лекционные плакаты и т. д. Кроме того, они согласовывают предварительный перечень заинтересованных лиц (перечисляя их не по именам, а по ролям), устанавливают сроки и получателей сводного отчета. Они также организуют снабжение специалистов по оценке архитектурной документацией – если, конечно, таковая существует и может оказаться полезной. Наконец, руководитель группы оценки объясняет руководителю проекта и архитектору, какую информацию им следует предоставить на первом этапе, и при необходимости помогает им составить соответствующие презентации.

На первом и втором этапах проводится непосредственно оценка – все погружены в аналитические операции. К началу этих этапов участники группы оценки должны ознакомиться с документацией по архитектуре, получить достаточное представление о системе, знать задействованные архитектурные методики и ориентироваться в первостепенных атрибутах качества. На первом этапе, намереваясь приступить к сбору и анализу информации, участники группы оценки встречаются с лицами, ответственными за проект (как правило, встреча длится весь день). На втором этапе к специалистам присоединяются заинтересованные в архитектуре лица, и в течение примерно двух дней они проводят аналитические мероприятия совместно.

Третий этап занимает доработка – группа оценки составляет в письменном виде и предоставляет получателям сводный отчет. По сути, участники занимаются самопроверкой и вносят в результаты своей работы разного рода коррективы. На заключительном совещании группы обсуждаются успехи и трудности. Участники изучают отчеты, выданные им на первом и втором этапах, и заслушивают выступление наблюдателя за процессом. По сути, они занимаются поиском путей усовершенствования по части исполнения своих ролей, с тем чтобы проводить последующие оценки с меньшими усилиями и с более высокой эффективностью. Действия, выполненные в период оценки участниками трех групп, тщательно регистрируются. По прошествии нескольких месяцев руководитель группы должен связаться с заказчиком оценки, для того чтобы оценить долгосрочные результаты ее проведения и сравнить издержки с выгодами.

Четыре этапа ATAM, их участники и приблизительный график представлены в таблице 4.2.

Таблица 4.2 - Этапы ATAM и их характеристики

Этап	Операции	Участники	Средняя продолжительность
0	Установление партнерских отношений и подготовка	Руководство группы оценки и основные ответственные за проект лица	Проходит в неформальной обстановке, согласно конкретной ситуации; может длиться несколько недель
1	Оценка	Группа оценки и ответственные за проект лица	1 день с последующим перерывом продолжительностью от 2 до 3 недель
2	Оценка (продолжение)	Группа оценки, ответственные за проект лица и заинтересованные лица	2 дня
3	Доработка	Группа оценки и заказчик оценки	1 неделя

4.4.2 Метод анализа стоимости и эффективности – количественный подход к принятию архитектурно-проектных решений

Метод анализа стоимости и эффективности (Cost Benefit Analysis Method, CBAM) базируется на методе ATAM и обеспечивает моделирование затрат и выгод, связанных с принятием архитектурно-проектных решений, и способствует их оптимизации. Методом CBAM оцениваются технологические и экономические факторы, а также сами архитектурные решения.

4.4.2.1. Контекст принятия решений

Все программные архитекторы и ответственные лица стремятся довести до максимума разницу между выгодами, полученными от системы, и стоимостью реализации ее проектного решения. Являясь логическим продолжением метода ATAM, CBAM основывается на его артефактах. Контекст CBAM изображен на рисунке 4.4.

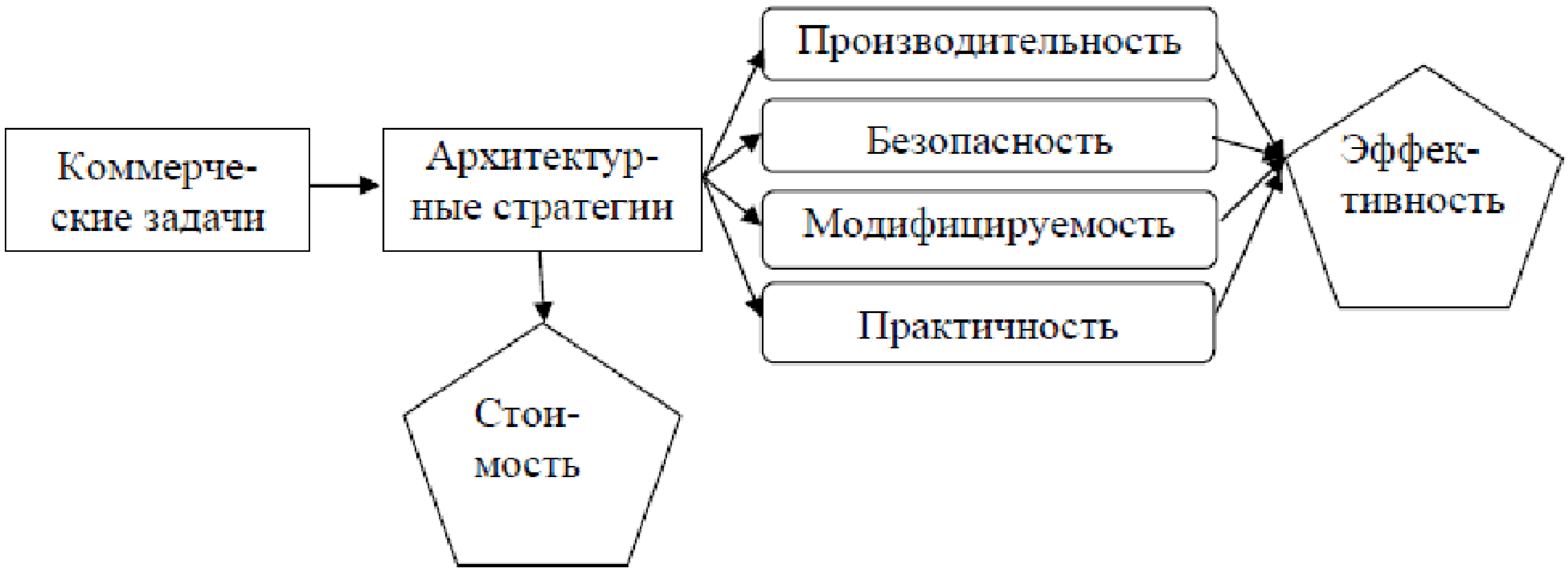


Рисунок 4.4 - Контекст метода анализа стоимости и эффективности (CBAM)

Поскольку архитектурные стратегии ограничиваются разнообразными техническими и экономическими факторами, стратегии, применяемые программными архитекторами и проектировщиками, должны быть поставлены в зависимость от коммерческих задач программной системы. Прямой экономический фактор – это стоимость реализации системы. Техническими факторами являются характеристики системы – другими словами, атрибуты качества. У атрибутов качества также есть экономический аспект – выгоды, получаемые от их реализации.

Как вы помните, по результатам оценки программной системы по методу ATAM в нашем распоряжении оказался ряд документированных артефактов.

Описание коммерческих задач, определяющих успешность системы.

Набор архитектурных представлений, документирующих существующую или предложенную архитектуру.

Дерево полезности, выражающее декомпозицию задач, которые заинтересованные лица ставят перед архитектурой, — от обобщенных формулировок атрибутов качества до конкретных сценариев.

- Ряд выявленных рисков.
- Ряд точек чувствительности (архитектурных решений, которые оказывают влияние на отдельный показатель атрибута качества).
- Ряд точек компромиссов (архитектурных решений, которые воздействуют сразу на несколько показателей атрибута качества, причем на одни положительно, а на другие отрицательно).

ATAM помогает выявить ряд основных архитектурных решений, значимых в контексте сформулированных заинтересованными лицами сценариев атрибутов качества. Эти решения приводят к реакции со стороны атрибутов качества, точнее говоря, отдельных уровней

готовности, производительности, безопасности, практичности, модифицируемости и т. д. С другой стороны, каждое архитектурное решение связано с определенными издержками (стоимостью). К примеру, достижение желаемого уровня готовности путем резервирования аппаратные средств подразумевает один вид издержек, а регистрация в файлах на диске контрольных точек – другой. Эти архитектурные решения приводят к (предположительно разным) измеримым уровням готовности, имеющим определенную ценность для компании-разработчика системы. Возможно, ее руководство полагает, что заинтересованные лица заплатят большую сумму за систему с высокой готовностью (если, к примеру, это телефонный коммутатор или программное обеспечение для медицинского наблюдения), или боится погрязнуть в судебных разбирательствах в случае отказа системы (вполне разумно, если речь идет о программе управления антиблокировочной тормозной системой автомобиля).

ATAM обнаруживает архитектурные решения, принятые относительно рассматриваемой системы, и устанавливает их связь с коммерческими задачами и количественной мерой реакции атрибутов качества. Принимая эти данные на вооружение, CBAM помогает выявить связанные с такими решениями издержку и выгоды. Основываясь на этой информации, заинтересованные лица могут принять окончательные решения относительно резервирования аппаратной части введения контрольных точек и всех прочих тактик, направленных на повышение готовности системы. Вполне возможно, что они предпочтут сконцентрировать ресурсы, которые, как известно, ограниченны, на реализацию какого-то другого атрибута качества – например, на улучшение соотношения выгод и издержек за счет повышения производительности. Из-за ограниченности бюджета разработки и обновления системы каждое архитектурное решение по большому счету соревнуется за право на существование со всеми остальными.

Подобно финансовому консультанту, который никогда напрямую не укажет, во что вкладывать деньги, CBAM не заменяет собой решений, принимаемых заинтересованными лицами. Он лишь помогает им установить и документировать издержки и выгоды архитектурных инвестиций, осознать неопределенность этого «портфеля». На этой основе заинтересованные лица могут принимать рациональные решения, удовлетворяющие их потребности и сводящие к минимуму риски.

Метод CBAM исходит из предположения о том, что архитектурные стратегии (как совокупность архитектурных тактик) оказывают влияние на атрибуты качества системы, а те, в свою очередь, предоставляют заинтересованным лицам некоторые выгоды. Эти выгоды мы называем полезностью (utility). Любая архитектурная стратегия отличается той или иной полезностью для заинтересованных лиц. С другой стороны, есть издержки (стоимость) и время, которые необходимо потратить на реализацию этой стратегии. Отталкиваясь от этой информации, метод CBAM помогает заинтересованным лицам в процессе выбора архитектурных стратегий, характеризующихся максимальной прибылью на инвестированный капитал (return on investment, ROI), – другими словами, наиболее выгодных с точки зрения соотношения выгод и издержек.

4.4.2.2 Реализация CBAM

На рисуноке 4.5 изображена диаграмма процессов, составляющих основу CBAM. Первые четыре этапа на ней сопровождаются комментариями с указанием относительного числа рассматриваемых сценариев. Постепенно их число уменьшается, таким образом, заинтересованные лица сосредоточиваются на тех сценариях, которые, по их мнению, в контексте ROI возымеют наибольшее значение.

Этап 1. Критический анализ сценариев. Отбор трети от общего количества сценариев путем расстановки приоритетов	N сценариев
Этап 2. Уточнение сценариев. Определение уровней реакции атрибутов качества для наилучшего, наихудшего, текущего и желаемого вариантов сценариев	N/3 сценариев
Этап 3. Упорядочивание сценариев согласно приоритетам. Исключение половины сценариев	N/3 сценариев
Этап 4. Определение полезности для текущего и желаемого уровней каждого из сценариев	N/6 сценариев
Этап 5. Разработка для сценариев архитектурных стратегий и определение уровней реакции атрибутов качества	
Этап 6. Определение ожидаемой полезности архитектурной стратегии путем интерполяции	
Этап 7. Определение общей выгоды, полученной от архитектурной стратегии	
Этап 8. Отбор архитектурных стратегий на основе ROI и с учетом ограничений по затратам	
Этап 9. Наглядное подтверждение полученных результатов	

Рисунок 4.5 - Диаграмма процессов CBAM

Этап 1. Критический анализ сценариев. Критический анализ сценариев проводится в рамках ATAM; на этом же этапе заинтересованные лица могут формулировать новые сценарии. Приоритеты расставляются в соответствии с потенциалом сценариев в контексте выполнения коммерческих задач системы; по результатам этапа для дальнейшего рассмотрения отбирается треть от общего первоначального числа сценариев.

Этап 2. Уточнение сценариев. Уточнению подвергаются сценарии, отобранные по результатам первого этапа; основное внимание при этом уделяется их значениям стимула-реакции. Для каждого сценария устанавливаются наихудший, текущий, желаемый и наилучший уровни реакции атрибута качества.

Этап 3. Расстановка сценариев согласно приоритетам. Каждому заинтересованному лицу выделяется 100 голосов, которые он берется распределить между сценариями, исходя из их желаемых значений реакции. После подсчета голосов для дальнейшего анализа остается только половина сценариев. Сценарию с наивысшим рангом присваивается вес 1.0, и, отталкиваясь от него, значения веса устанавливаются для всех остальных сценариев.

Именно эти значения впоследствии задействуются при вычислении общей выгоды стратегии. Помимо прочего, на рассматриваемом этапе составляется список атрибутов качества, которые заинтересованные лица считают значимыми.

Этап 4. Установление полезности. Для сценариев, оставшихся после проведения этапа 3, определяются значения полезности всех уровней реакции (наихудшего, текущего, желаемого, наилучшего) атрибута качества.

Этап 5. Разработка для сценариев архитектурных стратегий и установление их желаемых уровней реакции атрибута качества. Разработка (или фиксация разработанных) архитектурных стратегий, ориентированных на реализацию выбранных сценариев, и определение ожидаемых уровней реакции атрибута качества. Учитывая то обстоятельство, что одна архитектурная стратегия иногда оказывает воздействие на несколько сценариев, расчеты

необходимо провести для каждого из затронутых сценариев.

Этап 6. Определение полезности ожидаемых реактивных уровней атрибута качества путем интерполяции. Исходя из установленных значений полезности (отраженных на кривой полезности) для рассматриваемой архитектурной стратегии определяется полезность желаемого уровня реакции атрибута качества. Эта операция проводится для каждого из перечисленных на этапе 3 значимых атрибутов качества.

Этап 7. Расчет общей выгоды, полученной от архитектурной стратегии. Значение полезности «текущего» уровня вычитается из желаемого уровня и нормализуется исходя из поданных на третьем этапе голосов. Суммируются выгоды, полученные от конкретной архитектурной стратегии, по всем сценариям и для всех значимых атрибутов качества.

Этап 8. Отбор архитектурных стратегий с учетом ROI, а также ограничений по стоимости и времени. Для каждой архитектурной стратегии определяются стоимостные и временные факторы. Значение ROI для стратегий определяется как отношение выгоды к издержкам. Архитектурные стратегии упорядочиваются по рангу согласно значениям ROI; впоследствии бюджет в первую очередь расходуется на высшие по рангу стратегии.

Этап 9. Интуитивное подтверждение результатов. Проверяется соответствие выбранных архитектурных стратегий коммерческим задачам компании. Если наблюдаются противоречия, ищем недосмотры во время проведения анализа. В случае если противоречия значительны, перечисленные этапы проводятся повторно.

План конспекта:

1. Планирование архитектуры.
2. Проектирование архитектуры.
3. Документирование программной архитектуры.
4. Методы анализа архитектуры.

Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-3	1-3	1-3	1-2

Тема самостоятельного изучения № 5
Технология MDA.

Вид деятельности студентов: самостоятельное изучение литературы
Итоговый продукт самостоятельной работы: конспект
Средства и технологии оценки: собеседование
Краткие теоретические сведения

5 ТЕХНОЛОГИЯ MDA
5.1 Использование архитектуры,управляемой моделью
5.1.1 Концепция архитектуры, управляемой моделью

Технология Model Driven Architecture (MDA) – архитектура, управляемая моделью, была разработана независимой некоммерческой организацией Object Management Group (OMG) – консорциум объектного управления. Она объединяет сотни компаний-производителей программного и аппаратного обеспечения.

В технологии MDA основным элементом проектирования считается модель.

Модель – это описание реальной системы, учитывающее определенные характеристики или аспекты моделируемого объекта, процесса или явления.

В конкретном проекте модель MDA представляет собой законченное и формализованное описание создаваемого программного продукта или его смысловой части. Высокая точность и непротиворечивость описания необходима для автоматизации процесса перевода модели в программный (обычно исходный) код.

Понятие «управление моделью» подразумевает прямое использование модели при любых действиях по проектированию, разработке и развертыванию системы. При внесении изменений в архитектуру приложения модель считается первичной. Пусть, например, требуется пополнить описание класса новым нулем или методом. В классическом (немодельном) подходе к разработке модификация описания класса выполнялась бы в исходном коде, ручным кодированием. В технологии MDA происходит модификация визуальной диаграммы, на которой класс представлен в виде графического элемента. На базе такой диаграммы исходный код с измененным описанием класса генерируется автоматически.

Под архитектурой, управляемой моделью, понимается полная и законченная архитектура приложения, целиком формируемая с помощью модели.

В жизненный цикл программы входит этап формализации и анализа требований заказчика. Идея ведения этого этапа с помощью визуальных моделей развивается уже много лет. Она реализована в виде визуальных высокоуровневых средств, понятных людям, слабо знакомым с технологиями программирования. Такие средства задают целостную внутреннюю архитектуру сложной информационной системы. Лучшие подобные решения поддерживают прямую и двустороннюю связь модели и программного кода. Из модели можно автоматически получить исходный код, а из исходного кода – визуальную модель. В результате удается плавно состыковать этап выработки и согласования требований с этапом кодирования и формирования исполнимого приложения.

В крупных проектах с моделью обычно работает не программист, а системный аналитик. Он отвлекается от мелких деталей реализации и перестает мыслить в терминах отдельных операторов языка программирования. Хороший проектировщик способен охватить и продумать структуру больших функциональных логических блоков и основные взаимосвязи между ними. Далее подобная модель детализируется и транслируется в исходный код на выбранном языке.

Архитектура (приложения), управляемая моделью, позволяет быстро создавать объемные программные системы разной функциональной направленности. У заказчика появляется возможность инвестировать основные средства не в конкретные технические решения, а, прежде всего, в реализацию требуемой логики приложения на модельном уровне, которая потом может почти автоматически развертываться на разных платформах. При этом достигается высокий уровень сохранности инвестиций. Ведь систему, созданную с помощью MDA, при появлении новых технологий можно адаптировать к ним с минимальными усилиями модификации модели и на высоком абстрактном уровне.

Технология MDA ориентирована на создание приложений, которые независимы от платформы, операционной системы и языков программирования. Она позволяет строить масштабируемые приложения из компонентов, которые могут использоваться повторно и многократно. Сам процесс разработки выполняется, как явствует из названия, под управлением модели.

Модель приложения – это взаимосвязанный набор визуальных диаграмм, наглядно описывающих внутреннюю структуру системы и принципы ее функционирования. Модель приложения не привязана к конкретному языку или конкретной среде программирования.

Визуальные диаграммы чаще всего строятся с помощью унифицированного языка моделирования UML (Unified Modeling Language). Его стандарт разработан группой OMG для задач объектно-ориентированного проектирования.

5.1.2 Модельные точки зрения и модели MDA

Когда разработчик взаимодействует с создаваемой системой в рамках подхода MDA, ему доступны три модельные точки зрения на систему.

Точка зрения, независимая от вычислительных особенностей Computation Independent Viewpoint (CIV), нацелена на анализ системной среды и системных требований. Логика работы пока что не рассматривается, возможно, она еще не определена. В CIV учитываются программные, аппаратные и другие технологические ограничения, которые не связаны напрямую с внутренней архитектурой разрабатываемой программы.

Точка зрения, независимая от программно-аппаратной платформы Platform Independent Viewpoint (PIV), задает конкретные функциональные элементы системы, не зависящие от платформы. В ее рамках используются языки моделирования наподобие UML.

Точка зрения, зависящая от платформы Platform Specific Viewpoint (PSV), задает детали реализации, зависящие от используемых платформ и языков программирования. Она сочетает платформенно-независимую логику с дополнительными особенностями, вызванными использованием конкрет- ной платформы или системы.

Каждая из этих точек зрения предлагает разные средства построения соответствующих моделей. Бизнес-аналитики, незнакомые с особенностями программной разработки, вырабатывают свои требования к системе на уровне CIV – с помощью так называемых вычислительно-независимых моделей (Computation Independent Model, CIM). Модели CIM задают основные требования к проекту и включают словари предметной области. Никакие технические характеристики в этих моделях не фиксируются.

Методология MDA описывает не столько моделирование, сколько метамоделирование, предусматривающее большую гибкость в конкретных, прикладных подходах к моделированию.

Метамоделирование – способ описания моделей, определяющий механизмы построения конкретных моделей программных систем с помощью базового словаря и набора ограничений, налагаемых на создаваемые модели. Сегодня вместо термина BMDA корпорация Borland применяет новый термин ECO. Технология ECO (Enterprise Core Objects) – ключевые корпоративные объекты, – реализующая концепцию MDA, стала наиболее важным улучшением последних версий среды Delphi. Она представлена в Delphi 2006 в виде третьей версии ECO III. Каждый компонент ECO представляет собой своеобразную программную «обертку» положений концепции MDA. Он является промежуточным слоем между средствами визуального проектирования программных моделей и их конкретной реализацией на языках программирования Delphi и C+.

Практика перевода требований заказчика напрямую в программный код на языке программирования почти всегда страдает неполноценностью. Заказчик мыслит одними понятиями, программист – совсем другими. Поэтому они не подозревают о множестве потенциально возможных проблем. Так, заказчику какие-то моменты автоматизации выбранных процессов могут казаться очевидными. Он о них и не упоминает, считая, что программа выполнит определенный набор действий сама. Программист, наоборот, подходит к созданию системы с узких технических позиций, программируя только очевидный набор функций, заданный в техническом задании, и редко учитывает все множество взаимосвязей между большим числом требований. В результате в середине проекта выясняется, что архитектуру решения невозможно пополнить новыми требованиями, которые заказчику кажутся простыми, естественными и подразумевавшимися с самого начала. В итоге приходится заново переделывать почти весь проект.

Технология ECO переводит процесс согласования требований на модельный уровень. Диаграммы UML обычно хорошо воспринимаются и заказчиком, и исполнителем. Поэтому, хотя использование визуального языка моделирования в MDA не обязательно, почти 100 % проектов MDA создаются с применением языка UML.

Построение готового приложения происходит в несколько этапов. Сначала строятся модели PIM, затем выполняется их перенос в модели PSM. Доступные па сегодняшний день продукты автоматизируют эти шаги на 50–70 %. Перенос моделей PSM в код конкретного языка программирования автоматизирован почти на 100 %.

Еще один подход, настоятельно рекомендуемый группой OMG при использовании подхода MDA, заключается в выделении логики приложения в отдельную, по возможности платформно-независимую структуру. Достигается это использованием языка объектных ограничений OCL, который сегодня считается частью языка UML. Команды OCL встраиваются непосредственно в модель UML, описывая и уточняя аспекты ее работы. Удобство OCL еще и в том, что его выражения напоминают естественный язык (предложения, записанные на английском). Это позволяет сохранять при построении моделей контакт с людьми (например, представителями заказчика), не являющимися специалистами в области программирования.

При реализации сложной логики далеко не всегда удается обойтись стандартными возможностями языков OCL и UML. Всевозможные манипуляции с наборами объектов, их фильтрация и сортировка решают до 70–90 % потребностей современных проектов. Такие задачи характерны для многих задач автоматизации деятельности организаций. Технология ECO позволяет быстро создавать работающие прототипы, но 10–30 % требований все равно приходится программировать вручную. Для этого используется исходный код на языке Delphi, в который

отображается модель UML, сформированная в среде Delphi. Можно выполнить и обратное преобразование: на основе измененного исходного текста получить готовую модель UML.

Физические объекты, структура которых описана с помощью языка UML, а особенности существования – с помощью языка OCL, функционируют в процессе работы приложения в специально выделенной области оперативной памяти, так называемом объектном пространстве. Технология ECO способна автоматически отображать модель не только в программный код, но и в код на языке SQL. Он генерируется для построения схемы базы данных, хранящей копию объектного пространства. Поддержка разных СУБД является в ECO одной из ключевых технологий. С ее помощью удается сохранять текущее состояние объектного пространства приложения в базе данных и затем, после перезапуска приложения, загружать его и продолжать работу с прерванного места.

Среда Delphi на основе подготовленной модели автоматически формирует схему базы данных с учетом версии и специфики конкретной СУБД.

Она автоматически создает в ней все таблицы, нужные для хранения объектного пространства и взаимосвязей между классами, даже если это требует введения дополнительных таблиц. По мере совершенствования модели следует периодически синхронизировать схему базы данных с внесенными в модель модификациями – для этого достаточно нажать одну кнопку. Такой процесс называется в технологии ECO эволюционным развитием базы данных.

Стыковка объектного пространства с пользовательским интерфейсом реализуется в ECO с помощью дескрипторов. Дескриптор ECO – это стыковочная точка (компонент промежуточного уровня), связывающая с помощью выражений OCL элементы модели UML (классы) с программным кодом и внутренней структурой обычного приложения Delphi. Дескрипторы задают способы формирования наборов объектов в объектном пространстве по критериям, заданным выражениями OCL. Они также обеспечивают доступ к ним элементов пользовательского интерфейса, а программному коду предлагают ссылки на объекты ECO.

5.2 Язык объектных ограничений OCL

Один из самых серьезных и справедливо критикуемых недостатков языка моделирования UML – предоставление разработчику только визуальных средств моделирования. Эти средства абстрактны и поэтому далеко не всегда способны точно и формально отразить тот или иной нюанс функционирования проектируемой системы. В результате проектировщик не находит достаточно выразительных средств, позволяющих уточнить ограничения на применение создаваемых структур, специфицировать способы и нюансы их внутреннего функционирования, ввести условия использования и так далее. Конечно, можно представить эти особенности программным кодом конкретной среды программирования. Однако язык UML нацелен, прежде всего, на создание платформно-независимых моделей. Поэтому применяемый для этого текстовый язык, связанный с моделями UML, как минимум, тоже должен быть независим от операционной системы и среды разработки.

В начале 1990-х гг. в корпорации IBM разрабатывалась методика объектно-ориентированного анализа и проектирования Syntropy. Она включала математический язык текстовых описаний элементов визуальных моделей, который оказался сложен для массового практического использования.

Впоследствии на его основе был создан язык объектных ограничений OCL (Object Constraint Language), который применялся тогда как язык моделирования. Сильной стороной языка OCL оказалась независимость от плат- формы реализации и легкая адаптация к разным средам программирования. Он активно применялся для описания особенностей объектных информаци- онных систем и в 1997 г. вошел в стандарт языка UML 1.1.

До появления языка OCL в системах визуального моделирования при уточнении ограничений на диаграммах UML применялись комментарии или рекомендации на естественном языке. Они нередко трактовались неоднозначно.

Язык OCL снял множество проблем при проектировании моделей UML. Он предоставил разработчику набор формальных средств взаимодействия с объектами создаваемого приложения. Со временем язык OCL перерос границы стандарта UML. Он нередко задействуется как универсальный и, главное, платформно-независимый язык описания бизнес-логики. Например, сценарии OCL можно выделять и настраивать в проекте независимо от

программного кода на конкретном языке.

Язык OCL не является языком программирования. Он предназначен для формального определения выражений, обрабатывающих объекты. Выражение OCL обычно привязано к определенному классу UML и задает множество экземпляров этого класса. Команды OCL выполняют также фильтрацию этого множества или, например, определяют число его элементов.

Язык OCL содержит развитые средства манипуляции над множествами объектов, поэтому он хорошо подходит для решения задач, где обычно применяется язык запросов к реляционным базам данных SQL. Язык OCL позволяет организовывать запросы с большей эффективностью и на более высоком модельном уровне абстракции, нежели язык SQL.

Выражение OCL фактически задает условие, которому должны удовлетворять все экземпляры соответствующего объекта UML. Результатом выполнения выражения OCL является множество объектов, удовлетворяющих этому условию. Условие, которое задается выражением OCL, называется инвариант. Оно представляет собой набор правил или шаблон, накладываемые на описание объекта. Значением инварианта является логическая величина (True или False). Выражения OCL встраиваются в модели UML или задаются в сопроводительной документации.

Язык OCL надежен и безопасен. Стандартом гарантируется, что любое его выражение будет вычислено без побочных эффектов и не окажет влияния на части модели (ни на какие ее состояния, значения или связи). Среда, вычисляющая выражение OCL, просто определяет результирующее значение, которое может впоследствии использоваться (хотя это и не обязательно). Сам язык OCL может применяться для реализации бизнес-логики, управления процессами и других задач только в качестве средства расчета выражений.

Привязка выражения OCL к объекту UML происходит через так называемое объявление контекста. В начале выражения задействуется наименование нужного класса объекта. Может также указываться ключевое слово self (ссылка на текущий объект). Оно допускается, если контекст вычисляемого выражения однозначен, очевиден и может быть связан с выражением автоматически.

5.2.1 Типы данных и операции OCL

В языке OCL используется четыре основных типа данных – значения могут быть целыми, вещественными (с плавающей запятой), логическими и строковыми. Над числами определены стандартные арифметические операции: сложение (знак +), вычитание (–), умножение (*), деление (/). Над значениями всех типов допускаются операции сравнения: меньше (<), больше (>), меньше или равно (<=), больше или равно (>=), не равно (<>), равно (=). Значения логических типов можно обрабатывать с помощью логических операций or (ИЛИ), and (И), not (НЕ), xor (исключающее ИЛИ), а также операции implies. Операция implies представляет собой особую форму логической операции И, результат которой зависит от порядка операндов: X implies Y – это не всегда то же самое, что Y implies X.

5.2.2 Инфиксная форма записи выражений OCL

В язык OCL входит ряд операций, которые являются своеобразными аналогами стандартных функций языка Delphi. Только записываются они не как функции Delphi (имя идентификатора и параметры в круглых скобках), а в так называемой инфиксной записи, которая напоминает форму записи обычных арифметических выражений.

Инфиксная запись выражения подразумевает размещение операндов по обе стороны знака операции и вычисление выражения слева направо без учета приоритетов операций.

Типичное выражение OCL представляет собой цепочку элементов, ссылок и вызовов операций, которые последовательно начиная с левого края выражения и до его правого края обрабатывают значение, полученное к моменту их вызова. Функциональные элементы и операции языка OCL применяются к аргументу, расположенному левее. Они записываются справа от него через точку. Вторым аргументом операции или вторым параметром функции OCL считается элемент, расположенный правее, его заключают в круглые скобки.

Так, для действительных чисел можно использовать операцию abs для определения модуля числа или операцию round для округления. Однако эта операция записывается не слева, как принято в языках программирования, а справа от обрабатываемого значения и через точку. Само

значение, соответственно, указывается слева от имени функции и считается ее первым аргументом. Так как второй параметр для данной операции не нужен, результат ее работы можно передать далее, поставив еще одну точку, вслед за которой можно записать очередную функцию.

Пусть, например, требуется округлить число 123,45 с помощью функции round. Соответствующее выражение OCL записывается следующим образом:

123.45.round

Получить модуль отрицательного числа –123 с помощью функции abs можно так:

–123.abs

Если операция OCL использует два аргумента, второй из них заключается в круглые скобки, как обычный параметр. Например, операция max по выбору максимального из значений x и y запишется следующим образом:

x.max(y)

Для логической обработки язык OCL предлагает несложную форму условного оператора. Такой оператор фактически представляет собой вычисляемое выражение и записывается следующим образом:

if условие then выражение 1 else выражение 2 endif

Если условие истинно, то вычисляется выражение 1, и результатом условного оператора считается его значение. В противном случае вычисляется выражение 2 и результатом считается уже его значение.

5.2.3 Последовательности доступа к объектам в языке OCL

Выражение OCL возвращает результат, который может быть: экземпляром одного из базовых типов; экземпляром одного из классов модели; метаинформацией (сведениями об определенном типе данных); набором или коллекцией экземпляров одного класса. Элементы такой коллекции могут быть упорядочены или неупорядочены. Внутри коллекции допускаются одинаковые экземпляры. Если к обычным, скалярным, значениям операции OCL применяются через точку (символ .), то к коллекциям – через комбинацию символов ->.

Для доступа к полям определенного объекта в языке OCL применяется запись, в которой сначала следует имя объекта, затем точка и имя поля объекта. Например, если имеется объект Computer (экземпляр класса Computer), а в этом классе описано поле Mouse, то обращение к полю Mouse записывается следующим образом:

Computer.Mouse

Если у поля Mouse (экземпляра класса Mouse), в свою очередь, имеются внутренние поля, то и к ним можно обращаться, продолжая запись через точку:

Computer.Mouse.Position

Поля объектов в выражении OCL могут быть не скалярными значениями, а коллекциями. Для того чтобы получить (выбрать) все существующие в программе на данный момент экземпляры определенного типа, используется стандартная операция allInstances. Например, все созданные в приложении экземпляры класса Computer можно получить следующим выражением:

Computer.allInstances

Результатом такого выражения будет не один объект, а коллекция, набор экземпляров класса Computer. Соответственно, запись

Computer.allInstances.Mouse

формирует коллекцию значений – экземпляров класса Mouse, входящих, в свою очередь, в состав класса Computer (физически – в текущее множество экземпляров этого класса).

5.2.4 Операции над коллекциями

Рассмотрим основные операции языка OCL, которые можно применять к коллекциям. Такие операции иногда называют итераторами. Они чаще всего задействуются в моделях ECO для описания логики работы программ и формирования наборов записей, отображаемых на экране. Эти операции записываются с использованием операции ->.

5.2.4.1 Стандартные операции

Для выполнения операций над коллекциями язык OCL предлагает ряд стандартных

функций:

- функция Size возвращает число элементов коллекции;
- функции MinValue, MaxValue возвращают, соответственно, минимальное или максимальное значение из набора;
- функция Sum подсчитывает сумму всех элементов набора;
- функция Count определяет число элементов коллекции, удовлетворяющих условию, заданному в качестве параметра. Например, выражение Count (Name='Сергей') вычислит число элементов коллекции, у которых в поле Name содержится строка 'Сергей';
- функции isEmpty, NotEmpty возвращают значение True, если, соответственно, в анализируемом наборе нет ни одного элемента или есть хотя бы один элемент.

Функция, применяемая к коллекции, может возвращать скалярный результат (единственное значение) или вектор (набор значений). Результаты работы функций MinValue, MaxValue, Count и некоторых других относятся к последнему типу. Результатом может быть коллекция из нескольких одинаковых значений, каждое из которых соответствует объекту исходной коллекции, отвечающему заданному условию. Если, например, в исходном наборе несколько минимальных величин с одинаковыми значениями, все они попадут в результирующую коллекцию. Чтобы гарантированно получить скалярный результат, надо применить к итоговому набору операцию First.

5.2.4.2 Операция Select

Операция Select позволяет выбрать в коллекции подмножество объектов, свойства которых отвечают заданному условию. Это условие указывается после ключевого слова Select в круглых скобках.

Пусть, например, в классе Computer (компьютер) имеется свойство Memory (объем оперативной памяти). Отбор всех объектов класса Computer, у которых объем оперативной памяти больше или равен 512 мегабайт (512 единицам, принятым в модели по умолчанию), выполняет следующее выражение OCL:

Computer.allInstances->Select(Memory >= 512)

В дальнейшем по отношению к результирующей коллекции (в инфиксном порядке) можно применять другие допустимые операции OCL, в том числе и саму операцию Select. Например, возможно такое обращение к полям текущего подмножества объектов:

Computer.allInstances->Select(Memory >= 512).Mouse

Это выражение отбирает объекты, представляющие компьютерные мыши (свойство Mouse), которыми оснащены компьютеры с оперативной памятью не менее 512 мегабайт.

5.2.4.3 Операция Reject

Операция Reject отбирает в выходную коллекцию только те объекты, которые не удовлетворяют заданному условию. По смыслу она противоположна операции Select. В примере

Computer.allInstances->Reject(Memory < 512)

будет получен тот же результат, что и при использовании выражения: Computer.allInstances->Select(Memory >= 512).

5.2.4.4 Выделение элементов коллекции

Для получения первого элемента в коллекции применяется операция First. Для получения последнего элемента в коллекции применяется операция Last. Выделение объекта по его номеру в коллекции выполняет операция At – номер нужного объекта задается в качестве параметра (нумерация начинается с единицы).

Например, следующее выражение получает последний элемент из коллекции компьютеров:

Computer.allInstances->last

Получить название (свойство Name) процессора (свойство Processor) первого элемента текущей коллекции компьютеров можно следующим образом: Computer.allInstances->first.Processor.Name

Получить объект, представляющий мышь пятого компьютера текущего набора, можно следующим образом:

Computer.allInstances->At(5).Mouse

5.2.4.5 Упорядочение набора

Применение операций выделения первого и последнего элементов не всегда имеет смысл, если исходная коллекция не упорядочена. Упорядочение наборов данных производится операциями OrderBy и OrderDescending. Они выполняют сортировку по заданному параметру, причем операция OrderDescending выполняет сортировку «в порядке убывания» – способом, противоположным OrderBy. Конкретный способ сортировки коллекций с помощью операции OrderBy определяется реализацией.

Например, отсортируем набор объектов-компьютеров по именам: Computer.allInstances->OrderBy(Name)

5.2.4.6 Логические итераторы

Проверка выполнения логического условия для всех элементов коллекции осуществляется с помощью итератора ForAll. Пусть, например, в классе Computer имеется свойство Mouse (отдельный класс), а у него, в свою очередь, имеется логическое свойство isMiddleButton (наличие средней кнопки мыши). Тогда выражение:

Computer.allInstances->ForAll(Mouse.IsMiddleButton) вернет значение True, если все без исключения компьютеры коллекции оснащены трехкнопочными мышами.

Итератор Exists, схожий по смыслу с итератором ForAll, возвращает значение True, если хотя бы один из элементов коллекции соответствует заданному условию:

Computer.allInstances->Exists(Mouse.IsMiddleButton)

5.2.4.7 Операции для работы со строками

Для работы со строками в языке OCL применяются следующие операции. Соединение двух строк в одну (символ +) может также задаваться ключевым словом concat. Строка-параметр присоединяется к исходному операнду справа. Например, следующее выражение к имени первого компьютера в коллекции прибавляет строку ': ноутбук':

Computer.allInstances->first.Name.Concat(': ноутбук')

Результатом выделения подстроки (ключевое слово Substring) является строка, выделенная из исходного операнда. Дополнительные параметры задают номера первого и последнего символов, включаемых в подстроку (нумерация начинается с единицы). Например, выражение

'компьютер'.Substring(2,4)

Вернет подстроку «омп».

Функции ToLower, ToUpper преобразуют все символы строки операнда, соответственно, к нижнему или верхнему регистру.

Функция strToint преобразует исходную строку в целочисленное значение.

5.2.4.8 Работа с датами

Даты представляются в языке OCL в формате #гггг-мм-дд, где гггг – четырехзначная запись года, мм – номер месяца, а дд – число в месяце. Схожим образом записываются и временные константы – в формате #чч:мм:сс. Здесь чч означает часы, мм – минуты, сс – секунды. Константы, записанные в таком виде, можно использовать в операциях сравнения. Например:

Computer.allInstances->Select(MyDate > #2006.01.01)

Users.allInstances->Select(ConnectTime = #23:59:00)

Чтобы проверить, лежит ли дата в указанном диапазоне, следует воспользоваться функцией inDateRange. Двумя ее параметрами выступают нижняя и верхняя границы дат.

5.3 Возможности технологии ECO

5.3.1 Введение в технологию ECO

Технология ECO – это реализация концепции архитектуры, управляемой моделью. Она позволяет полностью создать приложение с помощью языка UML.

Технология ECO включает в себя набор объектов моделирования .NET и работает только на платформе .NET. В ней используются диаграммы UML как на фазах начального проектирования приложения, так и на этапе эксплуатации программы. Это достигается за счет встраивания модели в приложение.

Почти весь процесс создания готового приложения в рамках проекта ECO осуществляется манипулированием визуальной моделью, представленной в виде диаграмм UML. При этом удается избежать ручной модификации исходного кода и минимизировать обращения к проектировщику пользовательского интерфейса – весь исходный код приложения автоматически генерируется на основе визуальной модели. Ручная модификация кода нужна обычно для тонкой настройки приложения на дополнительные требования пользователя.

Центральной концепцией программы ECO считается объектное пространство. Всевозможные аспекты создания и поведения объектов в этом пространстве задаются с помощью платформно-независимого языка объектных ограничений OCL.

Объектное (или модельное) пространство ECO – это область памяти, в которой существуют и взаимодействуют объекты ECO текущего проекта.

В системе Delphi 2006 поддерживается третья версия технологии ECO III. Она состыкована со средой моделирования Borland Together, выпускаемой как самостоятельный продукт. Технология ECO III позволяет проектировать не только статическую структуру приложения, иерархию классов, но и его поведение, программную логику. Для этого задействуются так называемые машины состояний (описывающие их диаграммы состояний появились в версии языка UML 2.0). В итоге почти вся разработка сложного приложения выполняется в дизайнере диаграмм UML, встроенном в систему Delphi. На базе созданных таким образом моделей автоматически генерируется полнофункциональный исходный код приложения. В случае ручной модификации исходных текстов структура модели также автоматически подстраивается под внесенные изменения. Такая двунаправленная технология синхронизации модели и кода получила в среде Delphi название LiveSource.

Технология ECO поддерживает не только этап построения приложения и модели ECO, но и этап выполнения программы. При запуске приложения ECO создается объектное пространство ECO, которое называется ECO Space. В этом пространстве хранится вся необходимая метаинформация о модели. В рамках объектного пространства происходит создание и уничтожение экземпляров элементов ECO. Содержимое пространства ECO можно автоматически связать с данными в файлах или базах данных.

Технология ECO хорошо подходит для построения масштабных корпоративных приложений. Она значительно сокращает время разработки приложений, в которых активно задействуются базы данных с множеством таблиц и сложными взаимосвязями и большое число экранных форм, создание которых вручную является трудоемкой рутинной задачей. В технологию ECO встроены средства поддержки реляционной модели баз данных. Они автоматически преобразуют модель UML в описание схемы базы данных, генерируют сценарии SQL, которые создают и модифицируют такие схемы, и выполняют другие необходимые операции.

5.3.2 Модель ECO

Модель может объединять несколько взглядов на объект, каждый из которых уточняет ту или иную особенность его существования. Модель компьютерной программы (в частности, модель ECO) отличается от физических и математических моделей тем, что на ее основе должен быть автоматически сгенерирован безошибочный и работоспособный исходный текст на заданном языке программирования. Это предъявляет повышенные требования к полноте модели ECO, ее точности и непротиворечивости.

Процесс подготовки модели удается упростить с помощью набора готовых объектов ECO. Они прозрачно обеспечивают взаимосвязь между программным кодом и элементами модели. В

результате проектирование и развитие архитектуры всего приложения существенно упрощается.

5.3.3 Пространство имен ECO

Познакомимся с организацией пространства имен ECO. В рамках иерархии классов Delphi все классы ECO входят в единое пространство имен Borland.Eco. Оно, в свою очередь, делится на шесть больших категорий.

Категория Borland.Eco.ObjectRepresentation содержит средства стандартного представления объекта ECO во внешних программах. Каждый из прикладных объектов ECO состоит из элементов (внутренние поля, методы). К каждому из этих элементов можно обращаться через стандартный интерфейс Element. Любой элемент объекта, доступный через этот интерфейс, может быть представлен в виде стандартного объекта .NET. Для этого задействуется его свойство AsObject.

Категория Borland.Eco.Handles содержит компоненты поддержки модельного пространства на этапе проектирования. Они связывают объекты ECO, доступные через интерфейсы пространства имен ObjectRepresentation, с элементами пользовательского интерфейса.

Категория Borland.Eco.UmIRt содержит средства доступа к модели UML в процессе работы программы. Компоненты ECO базируются на классах .NET, поэтому к ним можно обращаться универсальным способом, принятым в .NET. В частности, для этого задействуется метод AslObject, предоставляющий для доступа к объекту интерфейс IObject платформы .NET.

Категория Borland.Eco.Subscription содержит средства уведомления об изменении значений объектов ECO по технологии «издатель – подписчики».

Категория Borland.Eco.Persistence содержит средства автоматического сохранения содержимого объектного пространства ECO в файлах и базах данных.

Категория Borland.Eco.Services – это среда поддержки внутренних механизмов технологии ECO. Она занимается вычислением значений выражений OCL, вносит модификации в модель, контролирует ее целостность.

5.4 Разработка приложений на основе ECO

5.4.1 Этапы создания приложения по технологии ECO

Технология создания приложения ECO состоит из следующих этапов.

1. Формируется модель UML будущего приложения. Создаются классы, описываются их атрибуты и методы, настраиваются взаимосвязи.

2. В проект добавляются и настраиваются невизуальные компоненты, связывающие созданную модель UML с прикладной частью проекта.

3. Проектируется пользовательский интерфейс. Задействуются компоненты, обеспечивающие связь интерфейса с моделью UML.

4. Создается переносимая логика приложения на языке OCL. Элементы управления связываются с выражениями OCL для выполнения типичных стандартных действий (добавление, редактирование и удаление объектов: ECO).

5. Пространство ECO связывается с базой данных. В ней будет долговременно храниться его копия: все объекты ECO и связи между ними. Эта последовательность в ходе расширения функциональных возможностей приложения многократно повторяется. Но все вносимые в проект изменения, что принципиально важно, выполняются, начиная с модели, а не с исходного кода или пользовательского интерфейса.

5.4.2 Создание простого MDA-приложения

Рассмотрим пример создания простого MDA-приложения. Имеются две сущности: *Кафедра* и *Преподаватель*. Представим экземпляры этих сущностей в таблицах формы. В этих же таблицах будет возможность модификации значений отдельных полей представленных в них объектов (экземпляров *Кафедра* и *Преподаватель*).

Создается пустая заготовка приложения ECO командой File>New>Other. Выберем значок ECO WinForms Application во вкладке Delphi for .NET Projects (рисунок 5.1).

В диалоговом окне введем имя проекта (например, projDeanOffice) и его

местоположение (каталог). Нажмем кнопку *OK*.

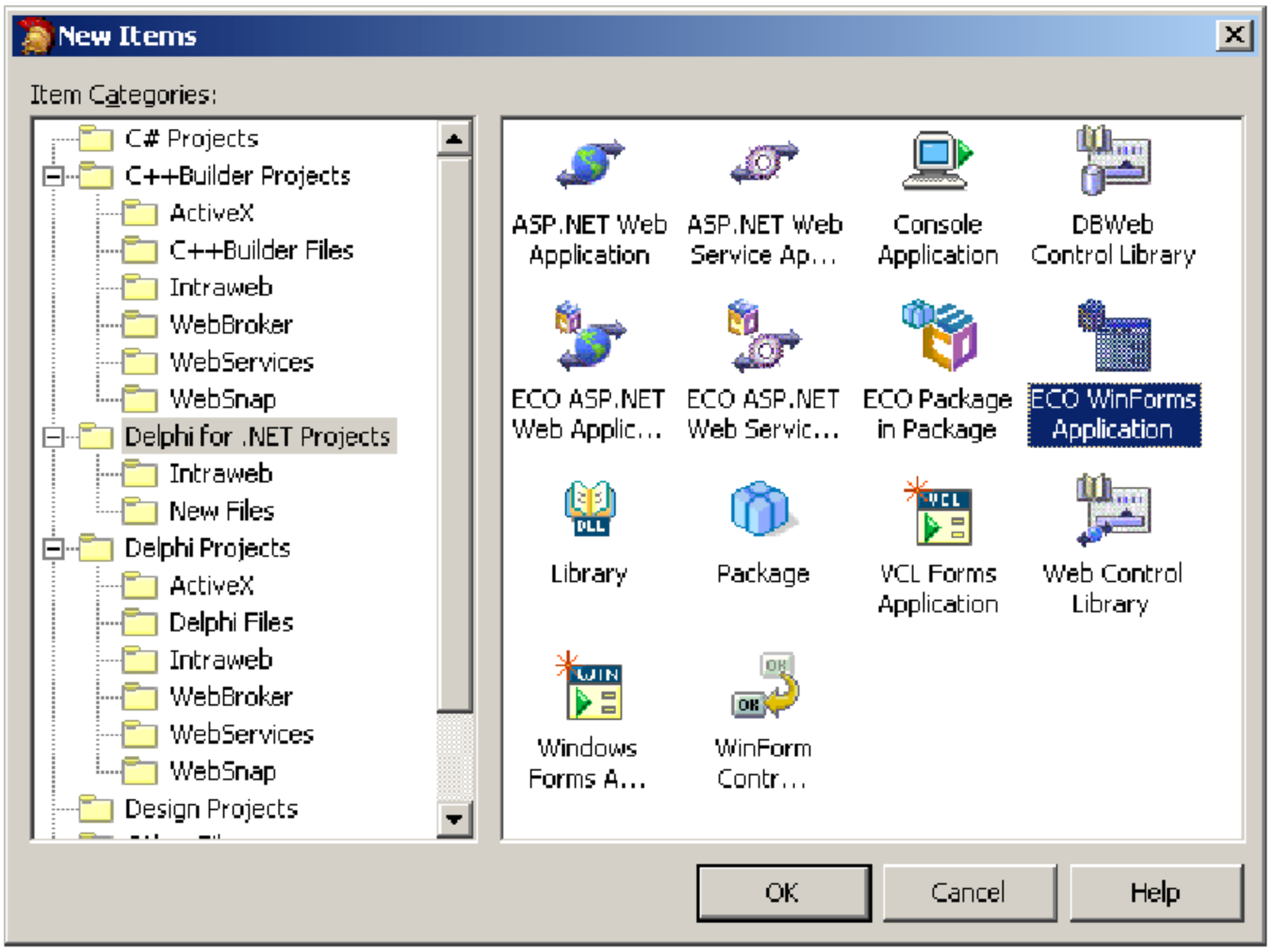


Рисунок 5.1 - Создание заготовки проекта ECO

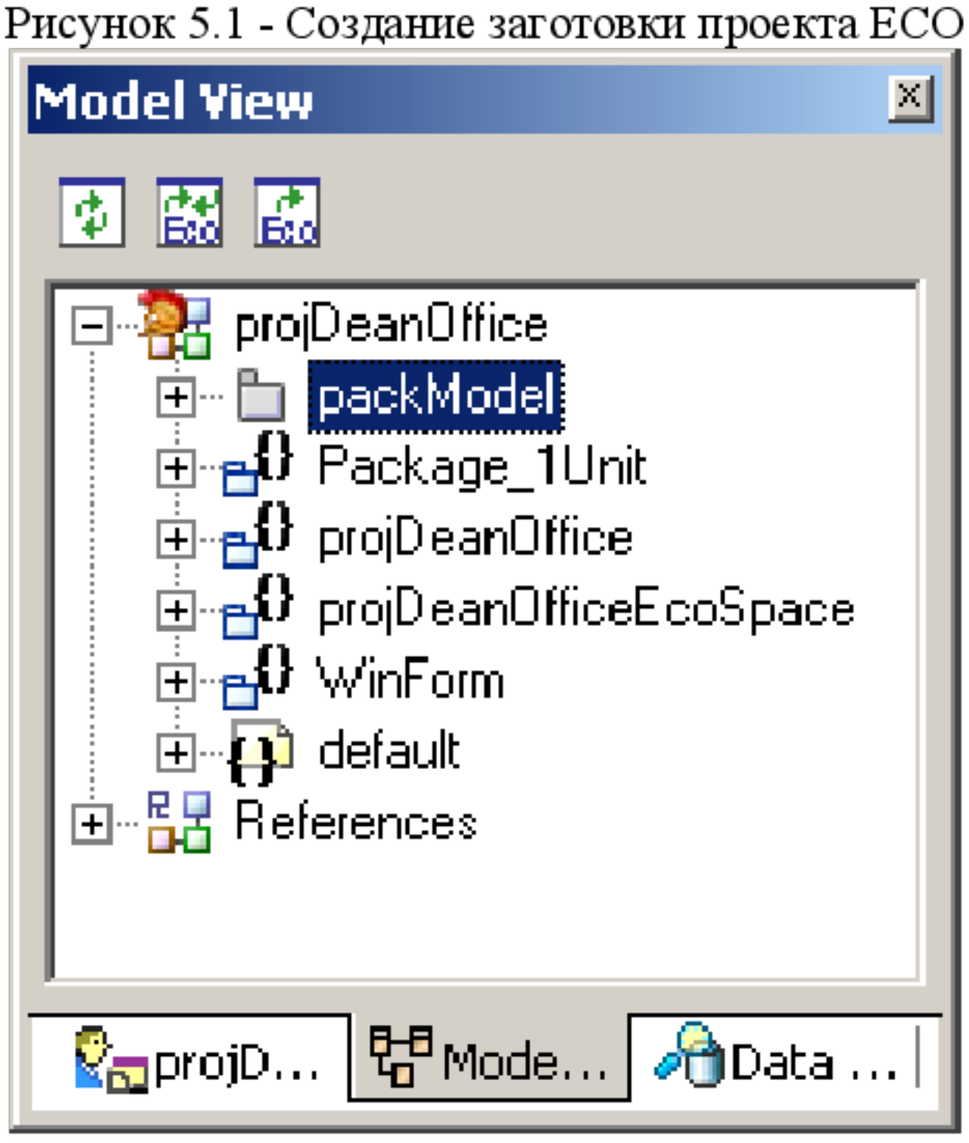


Рисунок 5.2 - Окно просмотра модели

Среда Delphi сформирует пустую заготовку приложения ECO и откроет окно *Проектировщика*. В нем расположена пустая начальная форма приложения. Структура автоматически созданной модели (пустой заготовки) доступна в окне просмотра модели, открываемом командой View>Model View либо выбором вкладки Model View в правом верхнем окне (рисунок 5.2). В этом окне в дополнение к самому проекту (projDeanOffice) и главной форме (Win-Form) можно увидеть еще несколько автоматически добавленных элементов. Среди них имеются следующие:

- элемент projDeanOfficeEcoSpace представляет объектное пространство ECO. Это основное хранилище, в котором располагаются экземпляры классов создаваемой модели во время работы программы. Это пространство также ответственно за хранение объектов ECO, например в базе данных или файле XML;

- элемент Package_1 представляет пакет классов UML, которые мы будем создавать.

Переименуем элемент Package_1 в удобное для нас название packModel.

5.4.2.1 Создание модели UML

Для создания модели дважды щелкнем мышью на строке packModel в окне просмотра модели. Откроется окно packModel [diagram] диаграммы классов ECO. Это окно напоминает окно построения обычных диаграмм классов UML. Только при работе с ним применяются элементы, специфичные именно для технологии ECO.

Разместим на диаграмме первый класс, отражающий сущность *Деканат*. Для этого выберем на палитре инструментов Tool Palette инструмент ECO Class в категории UML ECO Class Diagram и щелкнем в подходящей точке пространства моделирования. В ней появится графический элемент, изображающий класс.

Назовем класс clChair (*Кафедра*). Пробелы в имени класса ECO не допускаются.

Добавим атрибуты класса командой контекстного меню Add>Attribute. Создадим таким образом три атрибута: *Название кафедры*, *Ф. И. О. заведующего кафедрой*, *Ф. И. О. секретаря кафедры* – ChairName, ChairHeadSNP и ChairSecrSNP соответственно (рисунок 5.3). Для задания типа атрибута выделим нужный атрибут и в окне Properties установим необходимое значение поля Type в категории General. В данном случае все три атрибута имеют тип String.

Переименуем названия класса и атрибутов в понятные названия на русском языке. Для этого в свойстве Alias (категория General) класса и его атрибутов введем русскоязычные названия.

По аналогии добавим к модели еще один класс clLecturer (*Преподаватель*) с атрибутами LecturerSNP (*Ф. И. О. преподавателя*) и LectAcadDegree (*Ученая степень*) типа String. Переименуем элементы нового класса на диаграмме, изменив значения свойства Alias.

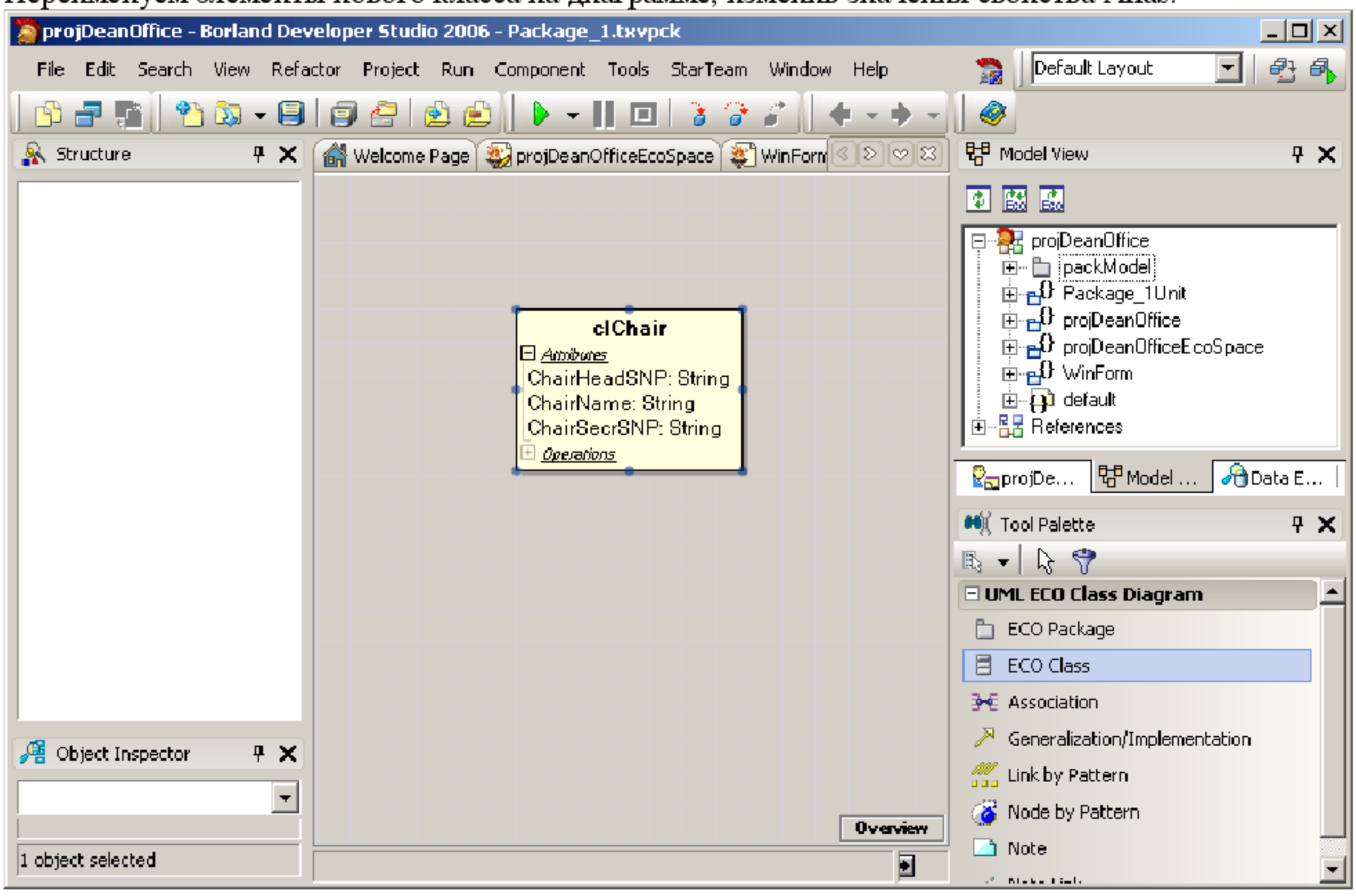


Рисунок 5.3 - Добавление класса clChair с его атрибутами

Настроим связи между созданными классами. Эта связь будет представлять отношение ассоциации. Выберем на палитре инструментов инструмент Generalization/Implementation. Щелкнем мышью на представлении класса *Кафедра*, протянем связь к классу *Преподаватель* и снова щелкнем мышью. В результате между двумя классами сформируется ассоциативное отношение.

Настроим мощность ассоциативного отношения. В нашем случае одному экземпляру

класса *Кафедра* соответствует множество экземпляров класса *Преподаватель* (от 1 и более). Для этого в окне Properties выделенной связи выберем категорию End1, соответствующую классу *Кафедра*, и в поле Multiplicity установим значение 1, а в категории End2 (для класса *Преподаватель*) этому же полю – значение 1..*. Теперь дадим сторонам связи назва- ния. В свойство Name для сторон End1 и End2 введем roleChair и roleLecturers соответственно (рисунок 5.4). То есть мы назвали роли каждого класса в ассоциативной связи.

На этом этапе создание простейшей модели закончено. Теперь надо организовать связь модели с пользовательским интерфейсом.

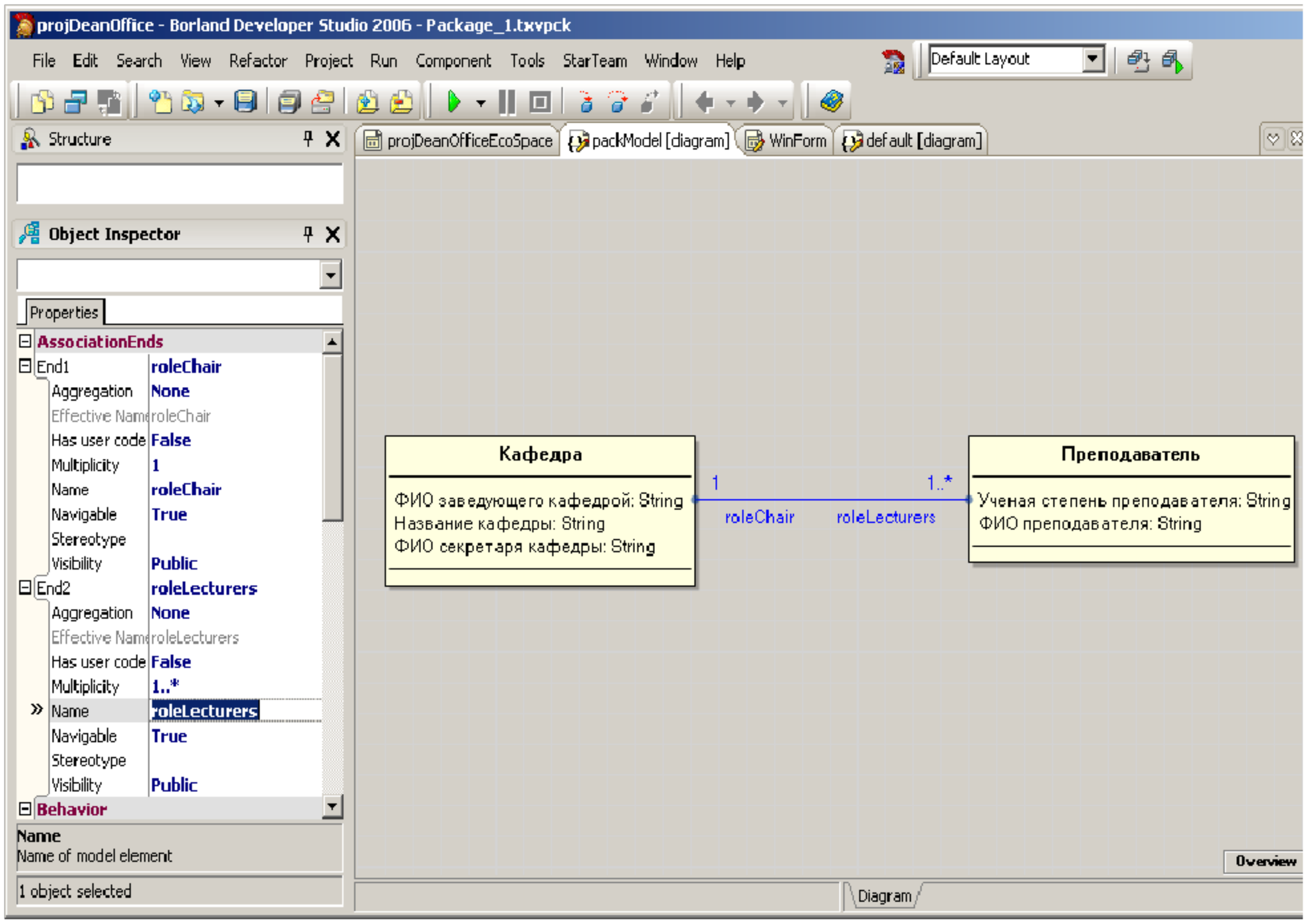


Рисунок 5.4 - Настройка связи между классами

5.4.2.2 Создание интерфейса

Перейдем к окну *Проектировщика* для главной формы проекта Win-Form (вкладка Design). Переименуем стандартное название главной формы.

Назовем ее wfMainForm, а сам класс TWinForm переименуем в TMainForm в свойстве Name категории Design. Свойство Text (категория Appearance) отвечает за название заголовка окна, введем в него строку, например *Главная форма*.

Для представления таблицы с объектами ECO на форме воспользуемся готовым компонентом DataGrid из категории палитры инструментов Data Controls. Поместим этот компонент на форму и дадим ему название dgChair (в свойстве Name). Эта таблица будет отвечать за представление экземпляров класса *Кафедра* (clChair). Исходно таблица должна быть пуста. В свойстве CaptionText (заголовок таблицы) введем название *Кафедры*.

Добавим еще одну таблицу dgLecturer, которая в готовом приложении будет отвечать за отображение экземпляров класса *Преподаватель* (clLecturer). В свойстве CaptionText введем название *Преподаватели*.

Для работы с таблицами в простом приложение потребуется пять операций: добавление и удаление данных в двух таблицах и сохранение копии ECO пространства из оперативной памяти в базу данных. Редактирование данных будет осуществляться непосредственно в полях таблиц. Добавим на форму пять кнопок – экземпляров класса Button (рисунок 5.5).

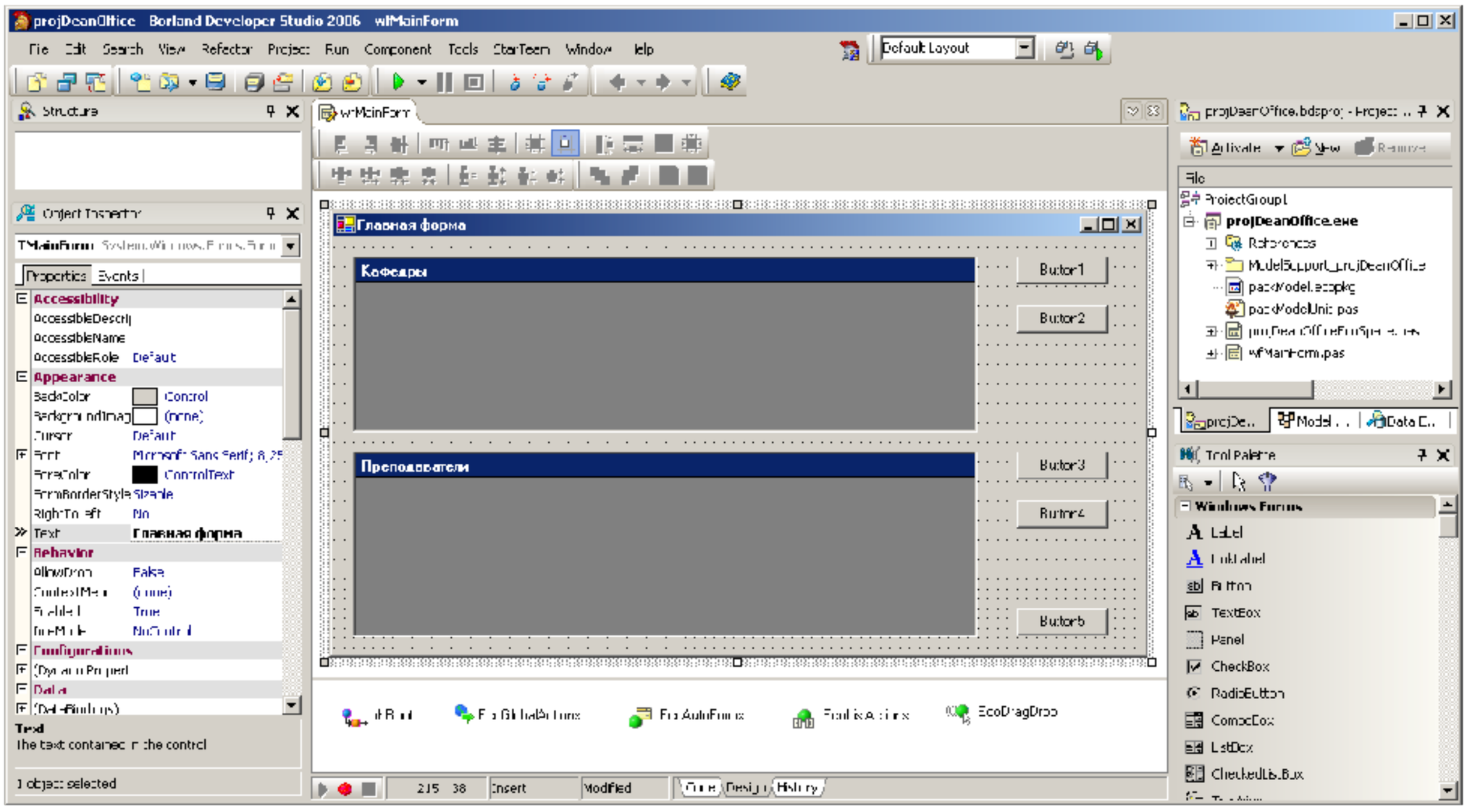


Рисунок 5.5 - Добавление компонентов пользовательского интерфейса

5.4.2.3 Связывание интерфейса с моделью

Связь пользовательского интерфейса с моделями ECO обычно происходит через компонент ExpressionHandle. Он доступен в категории палитры инструментов Enterprise Core Objects. Через подобные идентификаторы (де-скрипторы) организуется доступ к объектам модели и пространства ECO во время работы программы.

Идентификаторы ECO связываются друг с другом в цепочки. Во главе такой цепочки расположен корневой идентификатор. Он задает общий контекст работы (доступ к объектному пространству) для всех остальных идентификаторов ECO в программе.

Корневой идентификатор добавляется к проекту автоматически (по умолчанию он называется rhRoot). Все остальные идентификаторы ECO добавляются и настраиваются вручную, что связывает пространство ECO с элементами пользовательского интерфейса с помощью типовых механизмов связывания .NET.

Добавим к проекту компонент ExpressionHandle. Он отображается в нижней части окна Проектировщика, под формой, где уже имеется набор компонентов ECO. Назовем его ehChair, введем это имя в свойство Name категории Design.

В свойстве RootHandle этого дескриптора выберем значение rhRoot – ссылку на родительский, автоматически созданный корневой идентификатор.

Так задают место данного идентификатора в цепочке доступа к объектному пространству ECO.

В свойстве DataSource таблицы dgChair выберем имя ehChair в списке доступных идентификаторов ECO.

Настроим таблицу *Преподаватель* так, чтобы она отражала список преподавателей, принадлежащих выбранной кафедре в первой таблице. Для начала добавим в проект дескриптор CurrencyManagerHandle (из категории Enterprise Core Objects) и дадим ему имя cmhChair, так как этот компонент будет указывать на текущую строку таблицы *Кафедра*. Свяжем дескриптор cmhChair с таблицей dgChair через свойство BindingContext. Теперь он отслеживает выделенную строку этой таблицы.

Зададим ссылку на объект ehChair в свойстве RootHanler, определяющим корневой идентификатор ECO.

Добавим к проекту еще один компонент ExpressionHandle. Назовем его ehLecturer. Дескриптор ehLecturer будет обращаться к экземплярам класса *Преподаватель*.

Зададим ссылку на объект cmhChair в свойстве RootHanler.

Потребителем объектов, предоставляемых дескриптором ehLecturer, будет вторая

таблица. В ее свойстве DataSource выберем ссылку на объект ehLecturer.

Приступим к настройке элементов пользовательского интерфейса. Организуем с помощью визуальных средств Delphi создание новых экземпляров класса *Кафедра* и добавление их в таблицу. Выделим в окне *Проектировщика* кнопку Button1. В свойстве EcoListAction (Список стандартных действий ECO) выберем значение Add (*Добавить объект*).

Кнопка Button1 автоматически получила название Add. Переименуем кнопку. Для этого воспользуемся компонентом EcoListActions (он добавляется в проект по умолчанию и находится в нижней части окна *Проектировщика*), в его свойстве CaptionAdd введем *Добавить*.

Объект ECO, который мы хотим добавить к проекту (сделать доступным через элементы управления на форме), задается в свойстве RootHandle. В нем надо выбрать подходящий поставщик объектов ECO, в нашем случае – идентификатор ehChair.

Свяжем результат действия кнопки (созданный экземпляр класса *Кафедра*) с визуальным элементом, отображающим этот экземпляр, в нашем случае – с таблицей dgChair. Для этого в свойстве BindingContext (контекст связывания) выберем имя dgChair (рисунок 5.6).

Теперь добавим операцию удаления экземпляров класса *Кафедра*. Реализуем это действие по аналогии с операцией добавления. Выделим в окне *Проектировщика* кнопку Button2. В свойстве EcoListAction выберем значение Delete. Зададим идентификатор ehChair в свойстве RootHandle. Свяжем результат действия кнопки с визуальным элементом – таблицей dgChair. Для того в свойстве BindingContext у кнопки Delete выберем имя dgChair. Перейдем к компоненту EcoListActions и в его свойстве Caption Delete введем *Удалить*.

Настройка кнопки Button3 (будет отвечать за добавление экземпляра класса *Преподаватель*). Значения свойств кнопки: EcoListAction – Add; Roo- tHandle – ehLecturer; BindingContext – dgLecturer. Перейдем к компоненту EcoListActions и в его свойстве Caption Add введем *Добавить*.

Настройка кнопки Button4 (будет отвечать за удаление экземпляра класса *Преподаватель*). Значения свойств кнопки: EcoListAction – Delete; RootHandle – ehLecturer; BindingContext – dgLecturer. Перейдем к компоненту EcoListActions и в его свойстве Caption Delete введем *Удалить*.

Кнопка Button5 будет отвечать за обновление БД. Назовем ее Сохранить. В свойстве EcoAction (категория ECO|GUI) выберем UpdateDatabase.

Таким образом, по нажатию этой кнопки выполняется синхронизация содержимого пространства ECO в оперативной памяти и его копии в базе данных.

После каждого нажатия кнопки *Сохранить* содержимое таблицы сохраняется в БД. После нового запуска программы в таблицу пользовательского интерфейса автоматически загружается содержимое модели, сохраненное при последнем выполнении команды UpdateDatabase.

На данном этапе связывание интерфейса с моделью закончено (рисунок 5.7).

Действия, которые выполняет дескриптор в программе, определяются значением свойства Expression. В это свойство вводится выражение языка OCL, которое определяет схему создания объектов модели ECO.

Используем дескриптор ehChair для обращения к экземплярам класса *Кафедра*. Введем в свойство Expression объекта ehChair строку clChair.allInstances. Иначе это можно сделать с помощью редактора OCL выражений. Введенное выражение задает доступ ко всем текущим экземплярам класса *Кафедра* в объектном пространстве. В таблице, показанной на форме в окне Проектировщика, сразу же отобразятся поля класса *Кафедра* (Chair- HeadSNP, ChairSecrSNP и ChairName). Дескриптор ehChair становится поставщиком данных нашей модели ECO для первой таблицы.

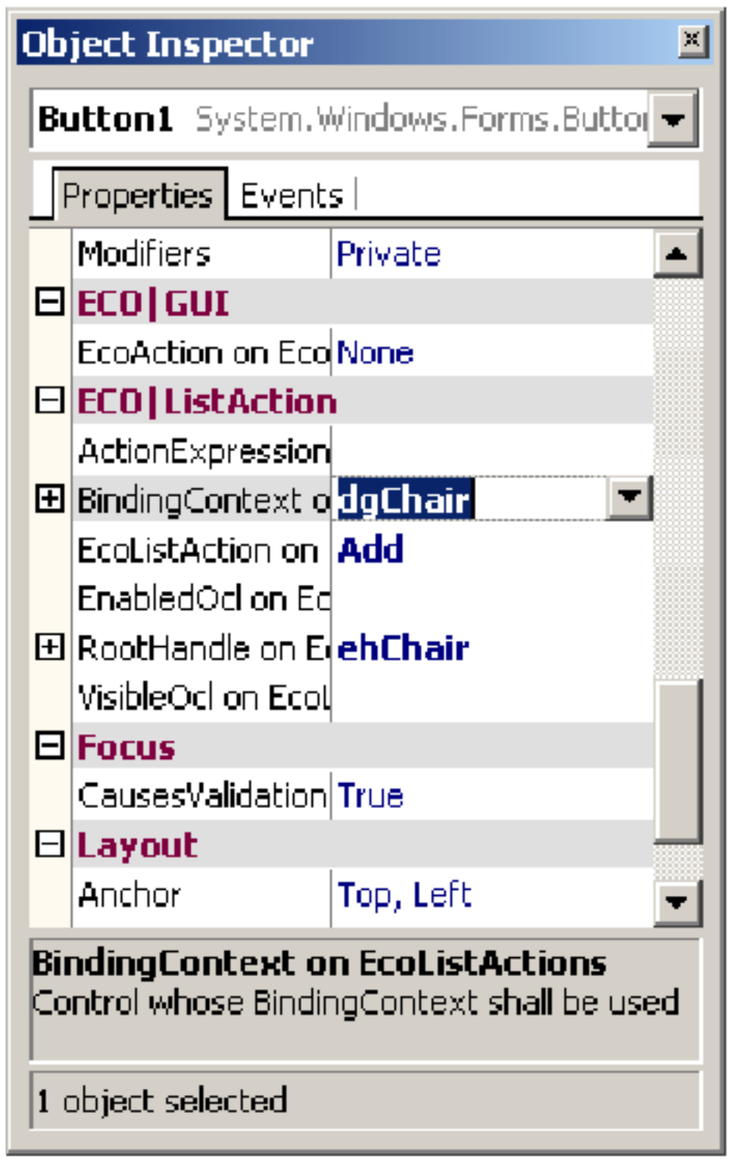


Рисунок 5.6 - Настройка кнопки добавления экземпляра класса Кафедра

5.4.2.4 Создание логики на OCL

В качестве выражения OCL объекта ehLecturer введем строку self.roleLecturers в свойство Expression. Здесь roleLecturers – это введенное нами при построении модели имя роли класса *Преподаватель* (clLecturer) в его ассоциативной связи с классом *Кафедра* (clChair). Это выражение определяет группу преподавателей, принадлежащих кафедре, которая выбрана в первой таблице. Дескриптор ehLecturer становится поставщиком данных для второй таблицы (рисунок 5.8).

Дадим заголовкам полей таблиц русскоязычные названия. Выберем на форме таблицу dgChair. Щелчком на кнопке с многоточием в строке свойства TableStyles откроем окно редактора коллекции стилей таблицы DataGridTableStyle Collection Editor. Создадим новый стиль таблицы, нажав на кнопку Add (рисунок 5.9).

Теперь настроим оставшиеся два элемента коллекции. Эти столбцы будут называться: *Ф. И. О. завкафедрой* и *Ф. И. О. секретаря кафедры*. Нажмем кнопку *ОК* в обоих открытых окнах и убедимся, что поля таблицы получили новые названия.

По аналогии настроим стиль таблицы *Преподаватели*. В ней отобразим два поля: LecturerSNP и LectAcadDegree. Назовем их *Ф. И. О. преподавателя* и Ученая степень.

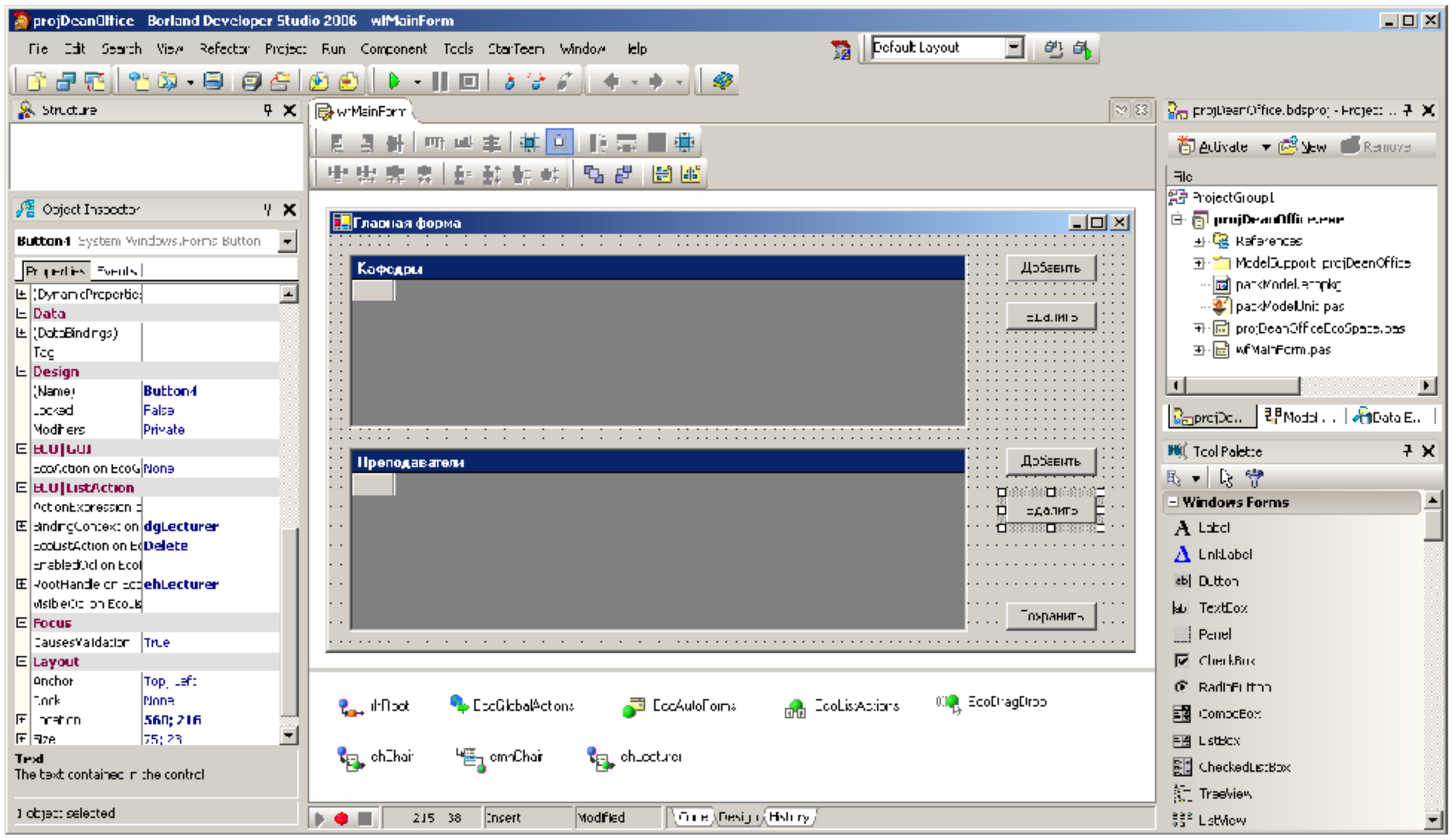


Рисунок 5.7 - Настроенный пользовательский интерфейс приложения

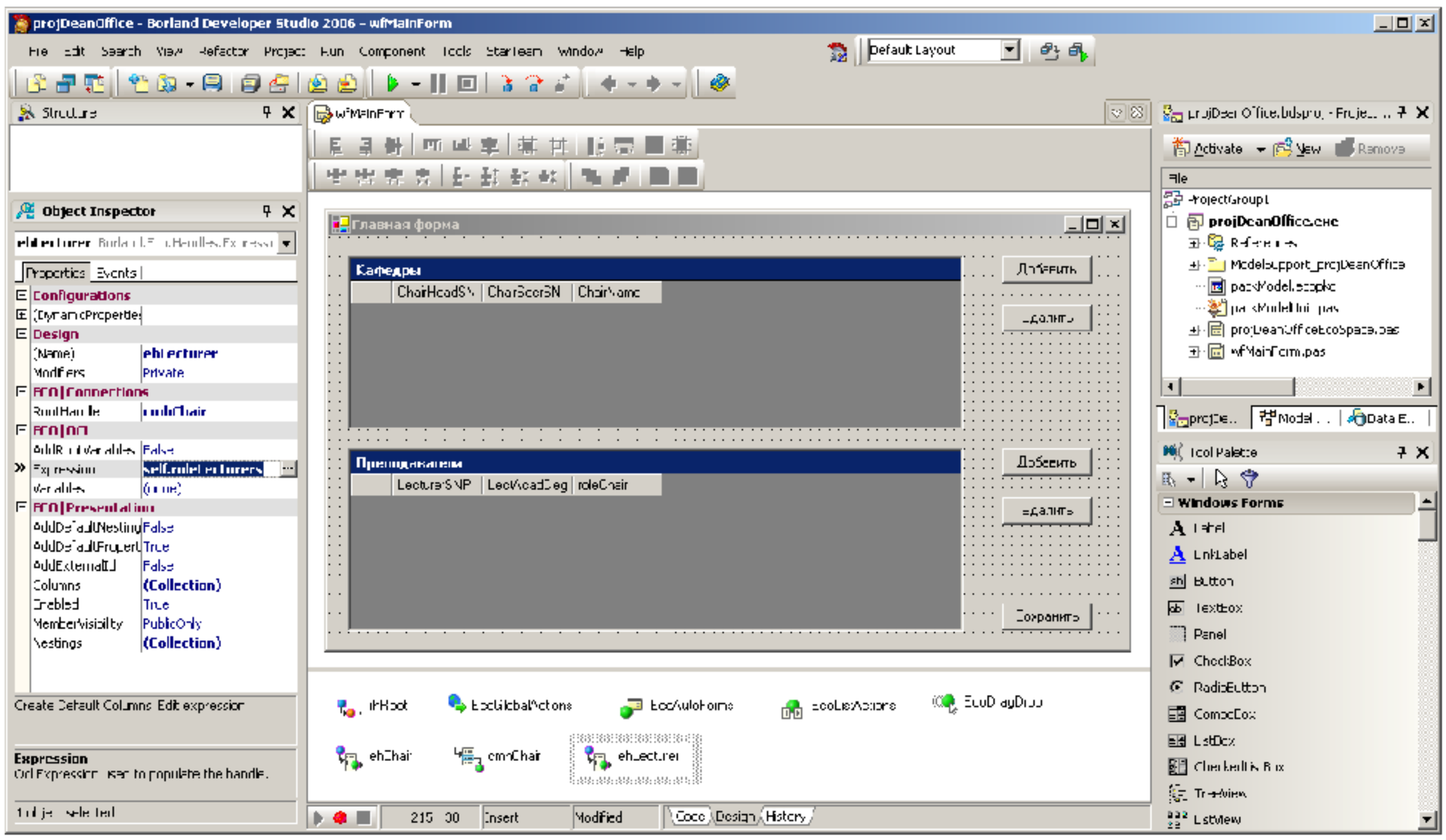


Рисунок 5.8 - Настройка OCL-выражений для взаимосвязанных таблиц

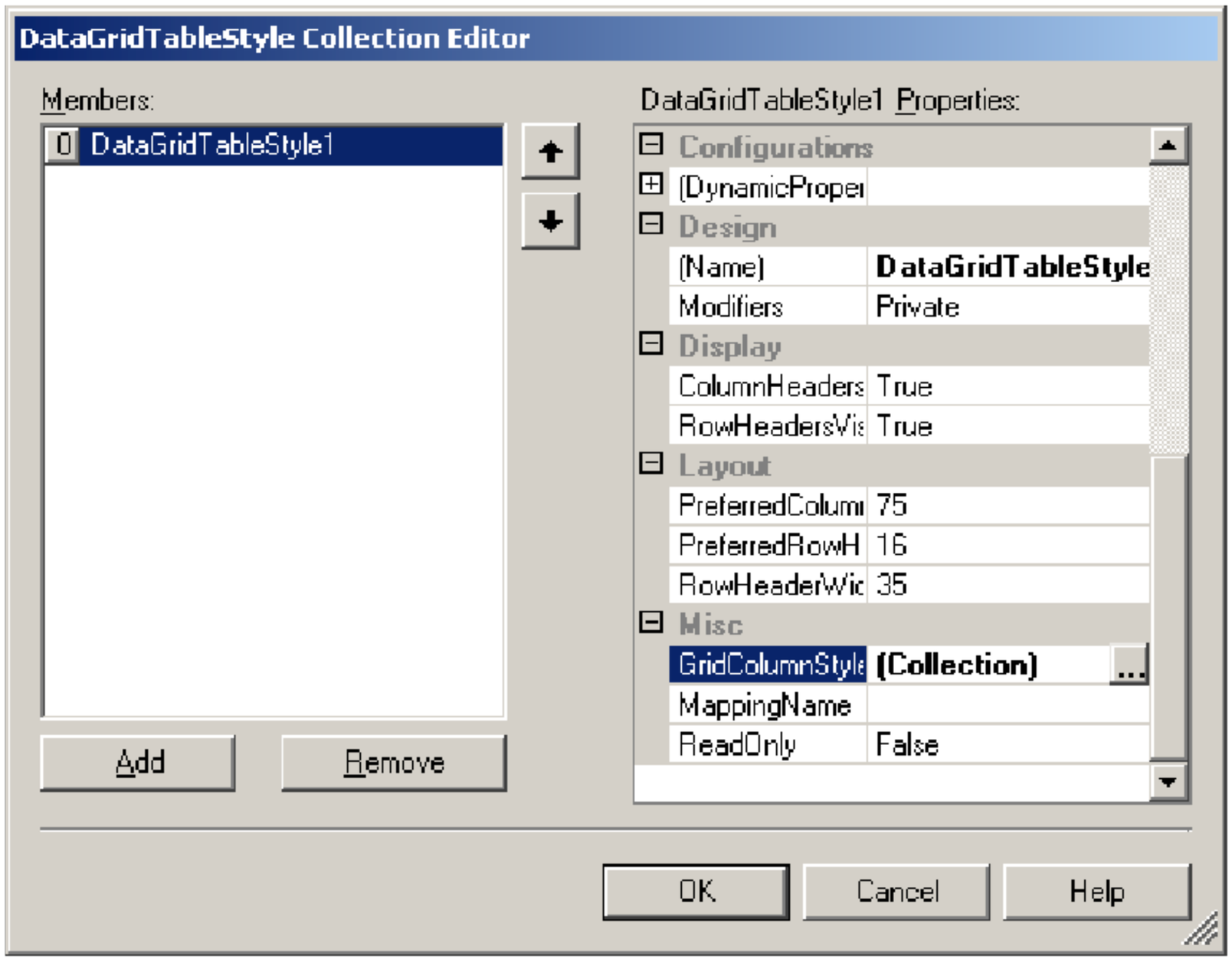


Рисунок 5.9 - Коллекция таблиц компонента gdChair

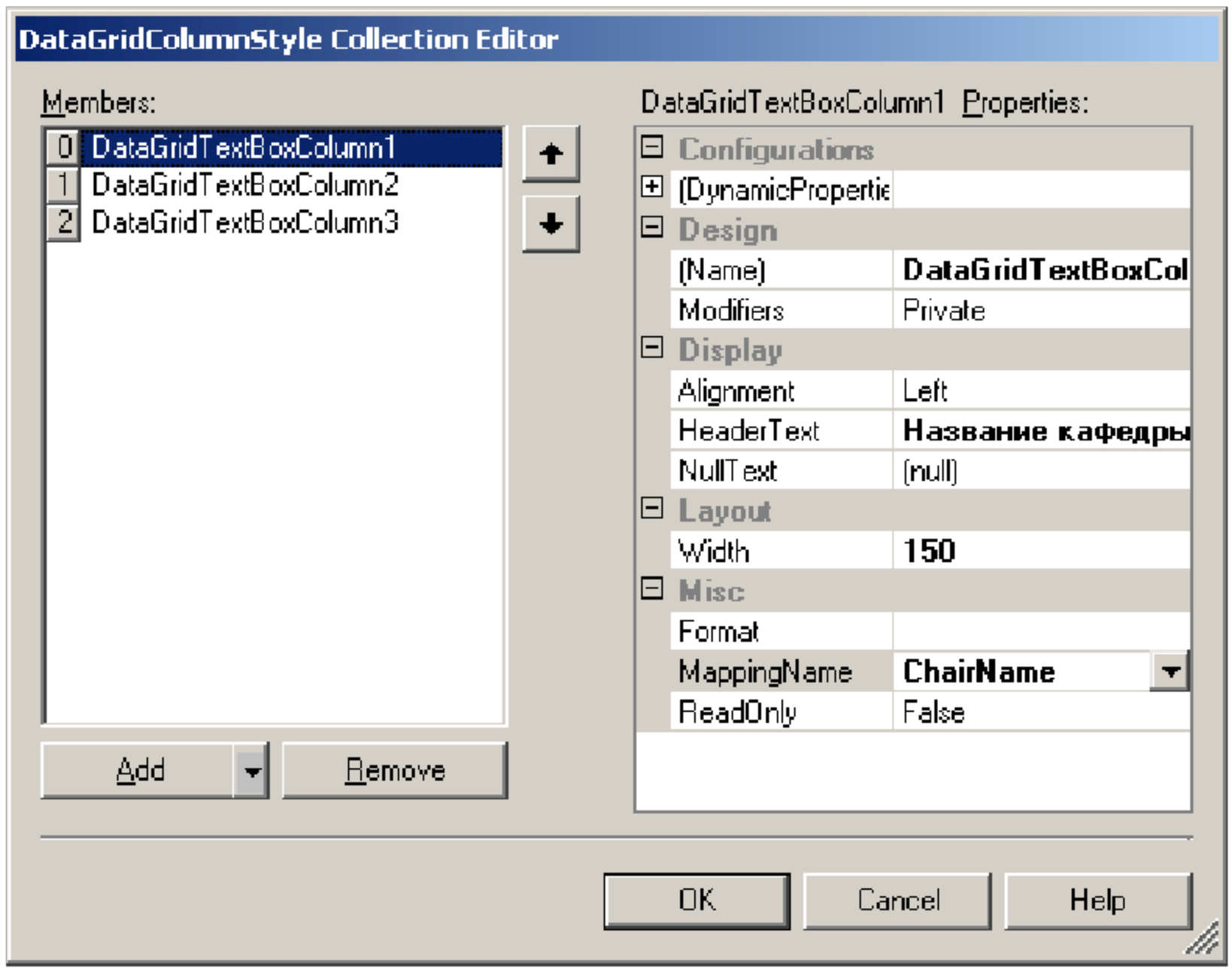


Рисунок 5.10 - Настройка элемента коллекции столбцов

Запустим программу и убедимся, что при нажатии кнопки *Добавить в таблице* автоматически формируются новые строки, отражающие содержимое новых экземпляров класса из объектного пространства ECO. В них можно ввести нужные значения.

План конспекта:

1. Использование архитектуры, управляемой моделью.
2. Язык объектных ограничений OCL.
3. Возможности технологии ECO.
4. Разработка приложений на основе ECO.

Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-3	1-3	1-3	1-2

5. ВОПРОСЫ К ЭКЗАМЕНУ

Вопросы к экзамену (3 семестр)

Вопросы (задача, задание) для проверки уровня обученности

Знать

1. Основные этапы развития технологии разработки. Этап 1 - «Стихийное» программирование. Этап 2 - Структурный подход к программированию. Этап 3 - Объектный подход к программированию.
2. Основные этапы развития технологии разработки. Этап 4 - Компонентный подход и CASE-технологии. Этап 5 - Разработка, ориентированная на архитектуру и CASE-технологии.
3. Эволюция моделей жизненного цикла программного обеспечения. Каскадная модель. Спиральная модель.
4. Эволюция моделей жизненного цикла программного обеспечения. Макетирование. Быстрая разработка приложений.
5. Эволюция моделей жизненного цикла программного обеспечения. Компонентно-ориентированная модель. XP-процесс.
6. Стандарты, регламентирующие процесс разработки программного обеспечения.
7. Системный анализ, основные понятия. Системные ресурсы.
8. Анализ проблемы и моделирование предметной области с использованием системного подхода.
9. Пять этапов, которые необходимо осуществить, при анализе проблемы с использованием системного подхода.
10. Методология ARIS. Организационная модель.
11. Методология ARIS. Диаграмма цепочки добавленного качества.
12. Методология ARIS. Модели eEPC.
13. Стандарты IDEF0 – IDEF3.
14. Методология описания бизнес-процессов IDEF3.
15. Методология функционального моделирования IDEF0.

Уметь,

Владеть

16. Методы определения требований. Интервьюирование. «Мозговой штурм» и отбор идей.
17. Методы определения требований. Совместная разработка приложений (JAD – Joint Application Design).
18. Методы определения требований. Раскадровка. Обыгрывание ролей.
19. Методы определения требований. CRC-карточки (Class – Responsibility – Collaboration, класс – обязанность – взаимодействие). Быстрое прототипирование.
20. Формализация требований. Метод вариантов использования и его применение.
21. Формализация требований. Псевдокод. Конечные автоматы.
22. Формализация требований. Графические деревья решений. Диаграммы деятельности.
23. Техническое задание. Общие сведения. Назначение и цели создания (развития) системы. Характеристики объекта автоматизации.

24. Техническое задание. Требования к системе. Состав и содержание работ по созданию (развитию) системы.
25. Техническое задание. Порядок контроля и приемки системы. Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие. Требования к документированию. Источники разработки.
26. Планирование архитектуры.
27. Программный процесс и архитектурно-экономический цикл.
28. Суть программной архитектуры.
29. Проектирование архитектуры. Атрибутный метод проектирования.
30. Проектирование архитектуры. Создание макета системы.
31. Документирование программной архитектуры. Варианты применения архитектурной документации.
32. Документирование программной архитектуры. Документирование представления (view).
33. Методы анализа архитектуры. Метод анализа компромиссных архитектурных решений – комплексный подход к оценке архитектуры.
34. Методы анализа архитектуры. Метод анализа стоимости и эффективности – количественный подход к принятию архитектурно-проектных решений.
35. Использование архитектуры, управляемой моделью. Концепция архитектуры, управляемой моделью.
36. Использование архитектуры, управляемой моделью. Модельные точки зрения и модели MDA.
37. Язык объектных ограничений OCL. Типы данных и операции OCL. Инфиксная форма записи выражений OCL. Последовательности доступа к объектам в языке OCL.
38. Язык объектных ограничений OCL. Основные операции языка OCL.
39. Возможности технологии ECO.
40. Разработка приложений на основе ECO.

6. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

Оценка «отлично» выставляется студенту, если теоретическое содержание курса освоено полностью, без пробелов; исчерпывающе, последовательно, четко и логически стройно излагает материал; свободно справляется с задачами, вопросами и другими видами применения знаний; использует в ответе дополнительный материал все предусмотренные программой задания выполнены, качество их выполнения оценено числом баллов, близким к максимальному; анализирует полученные результаты; проявляет самостоятельность при выполнении заданий.

Оценка «хорошо» выставляется студенту, если теоретическое содержание курса освоено полностью, необходимые практические компетенции в основном сформированы, все предусмотренные программой обучения учебные задания выполнены, качество их выполнения достаточно высокое. Студент твердо знает материал, грамотно и по существу излагает его, не допуская существенных неточностей в ответе на вопрос.

Оценка «удовлетворительно» выставляется студенту, если теоретическое содержание курса освоено частично, но пробелы не носят существенного характера, большинство предусмотренных программой заданий выполнено, но в них имеются ошибки, при ответе на поставленный вопрос студент допускает неточности, недостаточно правильные формулировки, наблюдаются нарушения логической последовательности в изложении программного материала.

Оценка «неудовлетворительно» выставляется студенту, если он не знает значительной части программного материала, допускает существенные ошибки, неуверенно, с большими затруднениями выполняет практические работы, необходимые практические компетенции не сформированы, большинство предусмотренных программой обучения учебных заданий не выполнено, качество их выполнения оценено числом баллов, близким к минимальному.

7. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

7.1. Рекомендуемая литература

7.1.1. Основная литература:

1. Битюцкая, Н. И. Разработка программных приложений : лабораторный практикум / Н. И. Битюцкая. — Ставрополь : Северо-Кавказский федеральный университет, 2015. — 140 с.

2. Абрамов, Г. В. Проектирование и разработка информационных систем : учебное пособие для СПО / Г. В. Абрамов, И. Е. Медведкова, Л. А. Коробова. — Саратов : Профобразование, 2020. — 169 с. — ISBN 978-5-4488-0730-5. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/88888.html (дата обращения: 10.04.2022). — Режим доступа: для авторизир. пользователей. - DOI: https://doi.org/10.23682/88888

7.1.2. Перечень дополнительной литературы:

1. Баженова, И. Ю. Основы проектирования приложений баз данных : учебное пособие / И. Ю. Баженова. — 3-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. — 324 с. — ISBN 978-5-4497-0682-9. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/97569.html (дата обращения: 10.04.2022). — Режим доступа: для авторизир. пользователей

2. Гвозденко, Н. П. Разработка блок-схем алгоритмов : учебное пособие / Н. П. Гвозденко, С. А. Суслова. — Липецк : Липецкий государственный технический университет, ЭБС АСВ, 2021. — 59 с. — ISBN 978-5-00175-055-0. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/116169.html (дата обращения: 10.04.2022). — Режим доступа: для авторизир. Пользователей

3. Савельев, А. О. Проектирование и разработка веб-приложений на основе технологий Microsoft : учебное пособие / А. О. Савельев, А. А. Алексеев. — 4-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2022. — 418 с. — ISBN 978-5-4497-1650-7. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/120486.html (дата обращения: 10.04.2022). — Режим доступа: для авторизир. пользователей

7.2. Перечень учебно-методического обеспечения самостоятельной работы обучающихся по дисциплине (модулю)

1. Методические рекомендации по выполнению лабораторных работ по дисциплине " Технология разработки программного обеспечения "
2. Методические рекомендации по организации самостоятельной работы студентов по дисциплине " Технология разработки программного обеспечения "
1. Методические указания по выполнению курсовой работы по дисциплине «Технология разработки программного обеспечения».

7.3. Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины (модуля)

1. http://el.ncfu.ru/ – система управления обучением ФГАОУ ВО СКФУ. Дистанционная поддержка дисциплины «Цифровая грамотность и обработка данных»
2. http://www.un.org - Сайт ООН Информационно-коммуникационные технологии
3. http://www.intuit.ru – Интернет-Университет Компьютерных технологий.

8. Перечень информационных технологий, используемых при осуществлении образовательного процесса по дисциплине (модулю), включая перечень программного обеспечения и информационных справочных систем

При чтении лекций используется компьютерная техника, демонстрации презентационных мультимедийных материалов. На семинарских и практических занятиях студенты представляют презентации, подготовленные ими в часы самостоятельной работы.

Информационные справочные системы:

Информационно-справочные и информационно-правовые системы, используемые при изучении дисциплины:

1	КонсультантПлюс - http://www.consultant.ru/

Программное обеспечение:

1	Операционная система: Microsoft Windows 8: 2013-02(3000). Бессрочная лицензия. Договор № 01-эа/13 от 25.02.2013. Окончание бесплатной поддержки – 2023-01 ИЛИ Операционная система: Microsoft Windows 10: 2016-08(20), 2017-10(67), 2018-01(18), 2018-04(6), 2018-05(6), 2019-02(7). Бессрочная лицензия. Договоры № 27-эа/16 от 02.08.2016. и № 0321100021117000009_229123 от 10.10.2017. На текущий момент окончания поддержки не анонсировано.
2	Базовый пакет программ Microsoft Office (Word, Excel, PowerPoint). MicrosoftOfficeStandard 2013: договор № 01-эа/13 от 25.02.2013г., Лицензирование Microsoft Office https://support.microsoft.com/ru-ru/lifecycle/search/16674 Дата начала жизненного цикла 09.01.2013г.; набор обновлений Office 2013 Service Pack1 Дата начала жизненного цикла 25.02.2014г., Дата окончания основной фазы поддержки 10.04.2018; Дополнительная дата окончания поддержки 11.04.2023г.