минобрнауки россии

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Иркутский государственный университет»

(ФГБОУ ВО «ИГУ»)

Факультет сервиса и рекламы

Кафедра Прикладной информатики и документоведения

О.А. Николайчук

**Инструментальное средство объектно-ориентированного проектирования Enterprise Architect**

Методическое руководство

к лабораторному практикуму

Издательство

ООО «ЦентрНаучСервис»

Иркутск 2017

Составители: О.А. Николайчук

УДК 004.4'2

ББК

Инструментальное средство объектно-ориентированного проектирования Enterprise Architect: Методическое руководство к лабораторному практикуму / ИГУ; Сост.: О.А. Николайчук. – Иркутск, 2017. – 71 с.

Рассмотрены возможности инструментального средства объектно-ориентированного проектирования Enterprise Architect. Уделено внимание описанию основ объектно-ориентированного подхода и основным элементам языка UML. Подробно изложены принципы работы с Enterprise Architect при создании диаграмм Business Process, Requirements, Use Case, Domain, анализа пригодности, Sequence, Class, Activity, State, Component и Deployment. Использование полученных знаний позволит студентам получить навыки проектирования информационных систем на основе объектно-ориентированного подхода. Предназначено для студентов 4 курса факультета Сервиса и рекламы специальности «Прикладная информатика в экономике» для лабораторных практикумов по курсам «Проектирование информационных систем» и «Проектирование и управление жизненным циклом информационных систем».

Ил. 60.

Рецензент: А.Ю. Юрин, к.т.н. доцент Кафедры автоматизированных систем Института кибернетики им. Е.И. Попова Иркутского национального исследовательского технического университета

© Иркутский государственный университет, 2017

# Содержание

[1. Описание основ объектно-ориентированного подхода 4](#_Toc493076449)

[2. Описание основных диаграмм языка UML 5](#_Toc493076450)

[3. Создание проекта в системе Enterprise Architect 12](#_Toc493076451)

[3.1. Business Process Model 15](#_Toc493076452)

[3.2. Requirements Model 20](#_Toc493076453)

[3.3. Use Case Model 23](#_Toc493076454)

[3.4. Domain Model 28](#_Toc493076455)

[3.5. Диаграмма анализа пригодности 31](#_Toc493076456)

[3.6. Sequence Model 37](#_Toc493076457)

[3.7. Class Model 41](#_Toc493076458)

[3.8. Activity Model 49](#_Toc493076459)

[3.9. State Model 53](#_Toc493076460)

[3.10. Component Model 58](#_Toc493076461)

[3.11. Deployment Model 60](#_Toc493076462)

[4. Программная реализация 64](#_Toc493076463)

[5. Формирование отчетов 66](#_Toc493076464)

[6. Практическое задание по курсу 69](#_Toc493076465)

[Список литературы 70](#_Toc493076466)

[Приложение 71](#_Toc493076467)

# Описание основ объектно-ориентированного подхода

Приведем в качестве введения отрывок из книги Г. Буча об объектно-ориентированном проектировании.

Модели позволяют нам наглядно продемонстрировать желаемую структуру и поведение системы. Они также необходимы для визуализации и управления ее архитектурой. Модели помогают добиться лучшего понимания создаваемой нами системы, что зачастую приводит к ее упрощению и возможности повторного использования. Наконец, модели нужны для минимизации риска.

Если вы хотите соорудить собачью конуру, то можете приступить к работе, имея в наличии лишь кучу досок, горсть гвоздей, молоток, плоскогубцы и рулетку. Несколько часов работы после небольшого предварительного планирования - и вы, надо полагать, сколотите вполне приемлемую конуру, причем, скорее всего, без посторонней помощи. Если конура получится достаточно большой и не будет сильно протекать, собака останется довольна. В крайнем случае никогда не поздно начать все сначала - или приобрести менее капризного пса.

Если вам надо построить дом для своей семьи, вы, конечно, можете воспользоваться тем же набором инструментов, но времени на это уйдет значительно больше, и ваши домочадцы, надо полагать, окажутся более требовательными, чем собака. Если у вас нет особого опыта в области строительства, лучше тщательно все продумать перед тем, как забить первый гвоздь. Стоит, по меньшей мере, сделать хотя бы несколько эскизов внешнего вида будущей постройки. Без сомнения, вам нужно качественное жилье, удовлетворяющее запросам вашей семьи и не нарушающее местных строительных норм и правил, - а значит, придется сделать кое-какие чертежи с учетом назначения каждой комнаты и таких деталей, как освещение, отопление и водопровод. На основании этих планов вы сможете правильно рассчитать необходимое для работы время и выбрать подходящие стройматериалы. В принципе можно построить дом и самому, но гораздо выгоднее прибегнуть к помощи других людей, нанимая их для выполнения ключевых работ или покупая готовые детали. Коль скоро вы следуете плану и укладываетесь в смету, ваша семья будет довольна. Если же что-то не сладится, вряд ли стоит менять семью - лучше своевременно учесть пожелания родственников.

При разработке программного обеспечения тоже существует несколько подходов к моделированию. Важнейшие из них - алгоритмический и объектно-ориентированный.

Алгоритмический метод представляет традиционный подход к созданию программного обеспечения. Основным строительным блоком является процедура или функция, а внимание уделяется, прежде всего, вопросам передачи управления и декомпозиции больших алгоритмов на меньшие. Ничего плохого в этом нет, если не считать того, что системы не слишком легко адаптируются. При изменении требований или увеличении размера приложения (что происходит нередко) сопровождать их становится сложнее.

Наиболее современным подходом к разработке программного обеспечения является объектно-ориентированный. Здесь в качестве основного строительного блока выступает объект или класс. В самом общем смысле объект - это сущность, обычно извлекаемая из словаря предметной области или решения, а класс является описанием множества однотипных объектов. Каждый объект обладает идентичностью (его можно поименовать или как-то по-другому отличить от прочих объектов), состоянием (обычно с объектом бывают связаны некоторые данные) и поведением (с ним можно что-то делать или он сам может что-то делать с другими объектами).

Унифицированный язык моделирования (UML) является стандартным инструментом для создания "чертежей" программного обеспечения. С помощью UML можно визуализировать, специфицировать, конструировать и документировать артефакты программных систем.

# Описание основных диаграмм языка UML

UML – это графический язык. Общаясь на этом языке, разработчики однозначно понимают друг друга.

В UML выделяют девять типов диаграмм:

* диаграммы классов;
* диаграммы объектов;
* диаграммы прецедентов;
* диаграммы последовательностей;
* диаграммы кооперации;
* диаграммы состояний;
* диаграммы действий;
* диаграммы компонентов;
* диаграммы развертывания.

На **диаграмме классов** показывают классы, интерфейсы, объекты и кооперации, а также их отношения. При моделировании объектно-ориентированных систем этот тип диаграмм используют чаще всего. Диаграммы классов соответствуют статическому виду системы с точки зрения проектирования. Диаграммы классов, которые включают активные классы, соответствуют статическому виду системы с точки зрения процессов.

Моделирование системы предполагает идентификацию сущностей, важных стой или иной точки зрения. Эти сущности составляют словарь моделируемой системы. Например, если вы строите дом, то для вас как домовладельца будут иметь значение стены, двери, окна, встроенные шкафы и освещение. Каждая из названных сущностей отличается от других и характеризуется собственным набором свойств. У стен есть высота и ширина, они твердые и сплошные. У дверей также есть высота и ширина, они тоже сплошные, но, кроме того, снабжены особым механизмом, позволяющим им открываться в одну сторону. Окна похожи на двери, поскольку представляют собой проемы в стенах, но в остальном свойства указанных сущностей различаются. Окна обычно (хотя и не всегда) проектируют так, чтобы через них можно было смотреть, но не проходить.

Стены, двери и окна редко существуют сами по себе, поэтому необходимо решить, как они будут стыковаться друг с другом. Какие сущности вы выберете и какие отношения между ними решите установить, очевидно, определяется в зависимости от того, как вы собираетесь использовать комнаты в доме, как будете перемещаться между ними, а также от общего стиля и обстановки, которые входят в ваш замысел.

Строителей, обслуживающий персонал и жильцов интересуют разные вещи. Водопроводчики обратят внимание на трубы, краны и вентиляционные отверстия. Вас как домовладельца это особенно не касается, если не считать случаев, когда указанные элементы пересекаются с теми, которые попадают в ваше поле зрения, - вас волнует, например, где труба вмонтирована в пол и в каком месте крыши открывается вентиляция.

В языке UML все сущности подобного рода моделируются как классы. Класс - это абстракция сущностей, являющихся частью вашего словаря. Так, умозрительно вы можете считать, что "стена" - это класс объектов с некоторыми общими свойствами, такими как высота, длина, толщина, несущая это стена или нет, и т.д. При этом конкретные стены будут рассматриваться как отдельные экземпляры класса "стена", одним из которых является, например, "стена в юго-западной части моего кабинета".

У каждого класса должно быть *имя*, отличающее его от других классов. Имя класса - это текстовая строка. Взятое само по себе, оно называется простым именем; к составному имени спереди добавлено имя пакета, куда входит класс. Имя класса в объемлющем пакете должно быть уникальным.

На практике для именования класса используют одно или несколько коротких существительных, взятых из словаря моделируемой системы. Обычно каждое слово в имени класса пишется с заглавной буквы.

*Атрибут* - это именованное свойство класса, включающее описание множества значений, которые могут принимать экземпляры этого свойства. Класс может иметь любое число атрибутов или не иметь их вовсе. Атрибут представляет некоторое свойство моделируемой сущности, общее для всех объектов данного класса. Например, у любой стены есть высота, ширина и толщина; при моделировании клиентов можно задавать фамилию, адрес, номер телефона и дату рождения.

*Операцией* называется реализация услуги, которую можно запросить у любого объекта класса для воздействия на поведение. Иными словами, операция - это абстракция того, что позволено делать с объектом.

**Документ**

Заголовок

Регистрационный номер

Сохранить документ

Открыть документ

Имя

Атрибуты

Операции

Например, класс – документ; атрибуты – заголовок, регистрационный номер; операции – сохранить документ, открыть документ.

При построении абстракций вы довольно скоро обнаружите, что классы редко существуют автономно. Как правило, они разными способами взаимодействуют между собой. Это значит, что, моделируя систему, вы должны будете не только идентифицировать сущности, составляющие ее словарь, но и описать, как они соотносятся друг с другом.

**Отношением** (Relationship) называется связь между элементами.

Существует три вида *отношений*, особенно важных для объектно-ориентированного моделирования:

* *зависимости*, которые описывают существующие между классами отношения использования (включая отношения уточнения, трассировки и связывания);
* *обобщения*, связывающие обобщенные классы со специализированными;
* *ассоциации*, представляющие структурные отношения между объектами.

*Отношения зависимости* - это отношения использования. Например, трубы отопления зависят от нагревателя, подогревающего воду, которая в них течет. Чаще всего зависимости применяются при работе с классами, чтобы отразить в сигнатуре операции тот факт, что один класс использует другой в качестве аргумента

*Отношения обобщения* связывают общие классы со специализированными; здесь также применяются термины "субкласс/суперкласс" или "потомок/родитель". Например, "фонарь" - это окно с большой нераздвижной рамой, а патио - окно, раздвигающееся в стороны.

Отношения зависимости и обобщения могут имена, чаще они определяются в том случае, когда необходимо ссылаться на эти отношения.

*Отношения ассоциации* - структурные взаимосвязи между объектами: комнаты состоят из стен, в которые могут быть встроены двери и окна; иногда через стены проходят трубы отопления.

Ассоциации может быть присвоено имя, описывающее природу отношения.

Часто при моделировании бывает важно указать, сколько объектов может быть связано посредством одного экземпляра ассоциации. Это число называется кратностью (Multiplicity) роли ассоциации и записывается либо как выражение, значением которого является диапазон значений, либо в явном виде.

**Документ**

Заголовок

Регистрационный номер

Сохранить документ

Открыть документ

**Заявление**

**Письмо**

**Обобщение**

**Ассоциация**

**Зависимость**

**Адресат**

**Секретарь**

Например, отношение зависимости: документ – секретарь (секретарь имеет операцию – зарегистрировать документ); отношение обобщения: документ – заявление, документ – письмо; отношение ассоциации: писмо – адресат.

*Агрегирование*. Простая ассоциация между двумя классами отражает структурное отношение между равноправными сущностями, когда оба класса находятся на одном концептуальном уровне и ни один не является более важным, чем другой. Но иногда приходится моделировать отношение типа "часть/целое", в котором один из классов имеет более высокий ранг (целое) и состоит из нескольких меньших по рангу (частей).

Отношение такого типа называют агрегированием; оно причислено к отношениям типа "имеет" (с учетом того, что объект-целое имеет несколько объектов-частей). Агрегирование является частным случаем ассоциации и изображается в виде простой ассоциации с незакрашенным ромбом со стороны "целого".

**Компания**

**Отдел**

**часть**

**агрегирование**

**целое**

1

0..n

На **диаграмме объектов** представлены объекты и отношения между ними. Они являются статическими "фотографиями" экземпляров сущностей, показанных на диаграммах классов. Диаграммы объектов, как и диаграммы классов, относятся к статическому виду системы с точки зрения проектирования или процессов, но с расчетом на настоящую или макетную реализацию.

На **диаграмме прецедентов** представлены прецеденты и актеры (частный случай классов), а также отношения между ними. Диаграммы прецедентов относятся к статическому виду системы с точки зрения прецедентов использования. Они особенно важны при организации и моделировании поведения системы.

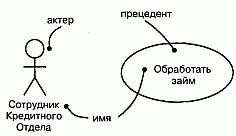
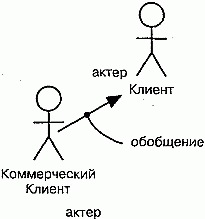
Чаще всего с помощью прецедентов моделируют поведение элемента: системы в целом, подсистемы или класса. При этом важно сконцентрироваться исключительно на том, что должен делать элемент, а не на том, как он это будет делать.

*Прецедентом* (Use case) называется описание множества последовательностей действий (включая варианты), выполняемых системой для того, чтобы актер мог получить определенный результат. Графически прецедент изображается в виде эллипса.

*Актер* представляет собой связное множество ролей, которые пользователи прецедентов исполняют во время взаимодействия с ними. Обычно актер представляет роль, которую в данной системе играет человек, аппаратное устройство или даже другая система. Например, если вы работаете в банке, то можете играть роль СотрудникКредит-ногоОтдела. Если в этом банке у вас имеется счет, вы играете роль Клиента. Таким образом, экземпляр актера представляет собой конкретную личность, взаимодействующую с системой определенным образом. Хотя вы и используете актеров в своих моделях, они не являются частью системы, так как существуют вне ее.

Моделирование поведения элемента осуществляется следующим образом:

* Идентифицируйте актеры, взаимодействующие с данным элементом. К числу актеров-кандидатов относятся группы, которые требуют определенного по ведения для выполнения своих задач либо необходимы, прямо или косвенно, для выполнения функций элемента.
* Организуйте актеры, выделив общие и специализированные роли.
* Для каждого актера рассмотрите основные пути его взаимодействия с элементом. Рассмотрите также взаимодействия, изменяющие состояние элемента или его окружения либо предполагающие реакцию на некоторое событие.
* Рассмотрите альтернативные (исключительные) способы взаимодействия актеров с элементом.
* Организуйте выявленное поведение в виде прецедентов, применяя отношения включения и расширения для выделения общего и исключительного по ведения.

Например, актер – секретарь, прецедент – обработать документ; сотрудник – секретарь.

Диаграммы прецедентов обычно включают в себя:

* прецеденты;
* актеры;
* отношения зависимости, обобщения и ассоциации.

Как и все остальные диаграммы, они могут содержать примечания и ограничения.

**Диаграммы последовательностей и кооперации** являются частными случаями диаграмм взаимодействия. На диаграммах взаимодействия представлены связи между объектами; показаны, в частности, сообщения, которыми объекты могут обмениваться). Диаграммы взаимодействия относятся к динамическому виду системы. При этом диаграммы последовательности отражают временную упорядоченность сообщений, а диаграммы кооперации - структурную организацию обменивающихся сообщениями объектов. Эти диаграммы являются изоморфными, то есть могут быть преобразованы друг в друга.

Чаще всего взаимодействия используют для моделирования потока управления, характеризующего поведение системы в целом, включая прецеденты (use case), поведение одного класса или отдельной операции. При этом классы и отношения между ними моделируют статические аспекты системы, а взаимодействия - ее динамические аспекты.

Моделируя взаимодействия, вы, по сути дела, описываете последовательности действий, выполняемых объектами из некоторой совокупности.

**На диаграммах состояний** (Statechart diagrams) представлен автомат, включающий в себя состояния, переходы, события и виды действий. Диаграммы состояний относятся к динамическому виду системы; особенно они важны при моделировании поведения интерфейса, класса или кооперации. Они акцентируют внимание на поведении объекта, зависящем от последовательности событий, что очень полезно для моделирования реактивных систем.

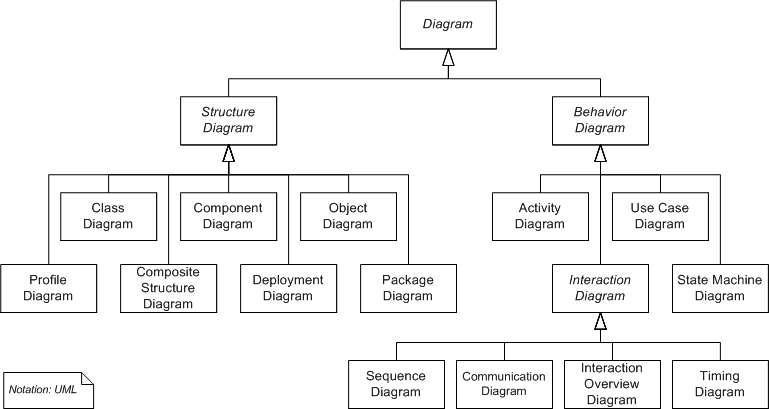
**Диаграмма деятельности** - это частный случай диаграммы состояний; на ней представлены переходы потока управления от одной деятельности к другой внутри системы. Диаграммы деятельности относятся к динамическому виду системы; они наиболее важны при моделировании ее функционирования и отражают поток управления между объектами.

На **диаграмме компонентов** представлена организация совокупности компонентов и существующие между ними зависимости. Диаграммы компонентов относятся к статическому виду системы с точки зрения реализации. Они могут быть соотнесены с диаграммами классов, так как компонент обычно отображается на один или несколько классов, интерфейсов или коопераций.

На **диаграмме развертывания** представлена конфигурация обрабатывающих узлов системы и размещенных в них компонентов. Диаграммы развертывания относятся к статическому виду архитектуры системы с точки зрения развертывания. Они связаны с диаграммами компонентов, поскольку в узле обычно размещаются один или несколько компонентов.

Здесь приведен неполный список диаграмм, применяемых в UML. Инструментальные средства позволяют генерировать и другие диаграммы, но девять перечисленных встречаются на практике чаще всего.

На рисунке представлена иерархия перечисленных диаграмм (рис. 0).

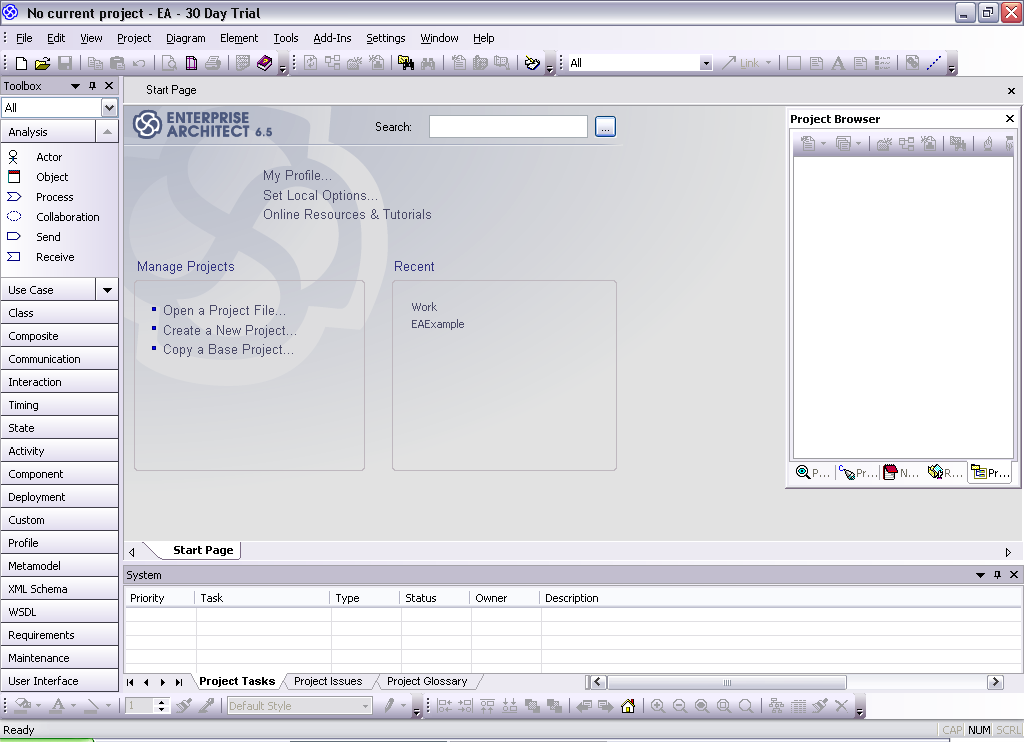


**Рис. 0.** Иерархия диграмм в нотации UML

# Создание проекта в системе Enterprise Architect

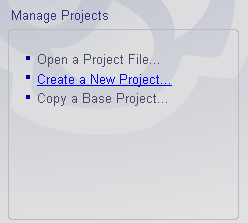
При запуске системы Enterprise Architect появляется основное окно (рис. 1), содержащее четыре основные области: Start Page, Toolbox, Project Browser, System.

Окно Start Page позволяет осуществить поиск по ключевым словам проекта, открыть проект, создать проект и др. Окно Toolbox предоставляет меню для выбора вида и компонентов диаграмм. Окно Project Browser позволяет управлять проектом (просматривать, редактировать, создавать элементы проекта). Окно System описывает события, выполняемые при создании проекта.



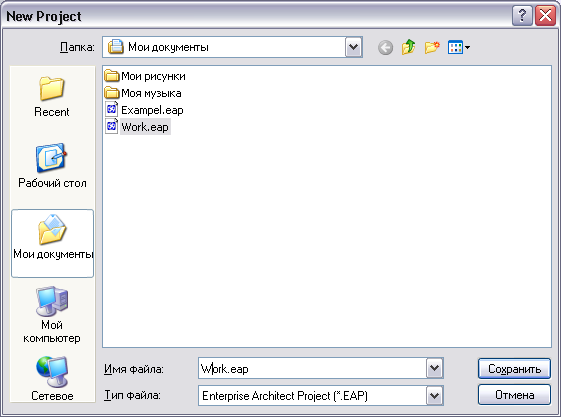
**Рис. 1.** Основное окно системы Enterprise Architect

Для создания проекта необходимо в области Manage Projects окна Start Page выбрать команду Create a New Project (рис. 2).



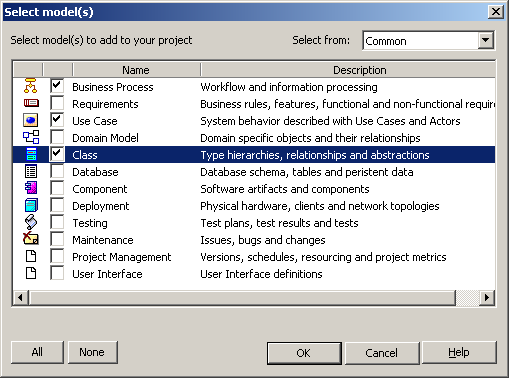
**Рис. 2.** Область Manage Projects окна Start Page

Система предлагает окно New Project для определения имени проекта. В рассматриваемом примере создадим проект с именем Work.eap (рис. 3). Проект сохраняется в файле с именем Work и расширением \*.eap.



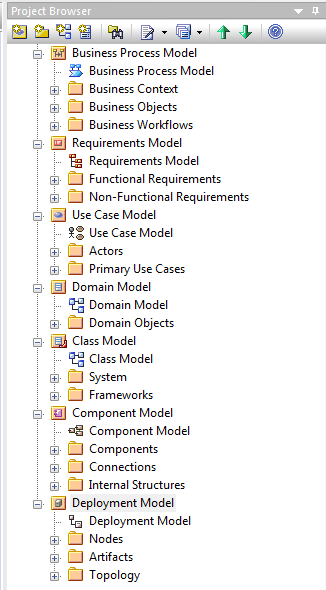
**Рис. 3.** Окно для задания имени проекта

Далее появляется окно, которое предлагает выбрать виды моделей (Select model(s) to add to your project) для использования в создаваемом проекте. Виды моделей перечислены в виде списка, где представлены: символ модели, имя модели (Name) и описание модели (Description). Для рассматриваемого проекта укажем следующие виды моделей, поставив галочки в списке напротив моделей Business Process, Requirements, Use Case, Domain Model, Class, Component и Deployment (рис. 4). Отметим, что на данном экране использованы символы:  – символ модели Business Process, – символ модели Requirements,  – символ модели Use Case;  – символ модели Domain Model, – символ модели Class;  – символ модели Component, – символ модели Deployment. Эти символы используются во всем проекте для обозначения указанных диаграмм.



**Рис. 4.** Окно выбора моделей для проекта

После выбора моделей система создаст проект со структурой, где предусмотрены папки (или пакеты) для хранения диаграмм Business Process (Business Process Model), Requirements ( Requirements Model), Use case ( Use Case Model), Domain Model (Domain Model), Class ( Class Model), Component ( Component Model) и Deployment ( Deployment Model) (рис. 5).



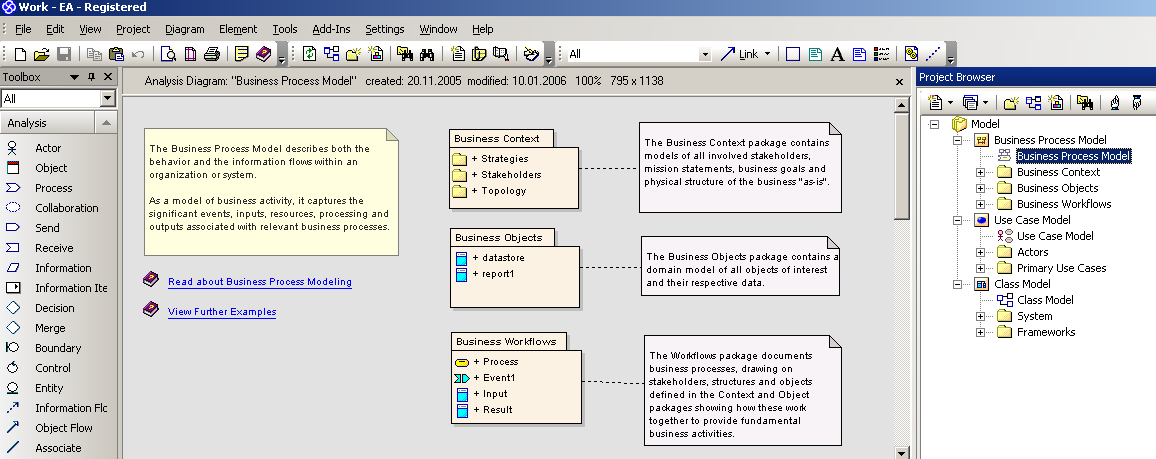
**Рис. 5.** Структура проекта, отображаемая в окне Project Browser

Рассмотрим содержание каждого из пакетов более подробно.

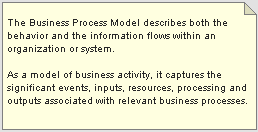
## Business Process Model

Пакет Business Process Model содержит:

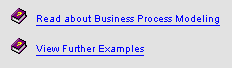
* Диаграмму  Business Process Model для краткого описания содержания основных пакетов диаграмм Business Context, Business Objects и Business Workflows (рис. 6), где представлены эти пакеты с содержанием по умолчанию, описание возможностей диаграммы (рис. 7), комментарии к пакетам, ссылки на справочную информацию (рис. 8).
* Пакет Business Context (бизнес-контекст) содержит модели всех заинтересованных сторон (Stakeholders), миссию, стратегию, бизнес-цели (Strategies) и физическую структуру бизнеса «как есть» (Topology).
* Пакет Business Objects содержит модели предметной области, все объекты, представляющие интерес и соответствующие данные.
* Пакет Business Workflows содержит описание рабочих процессов, пакетов документов, опираясь на заинтересованные стороны, структуры и объекты, определенные в Business Context и Business Objects, показывающие, как они работают вместе, чтобы обеспечить фундаментальные основы деятельности.



**Рис. 6.** Окно проекта, отображающее диаграмму  Business Process Model

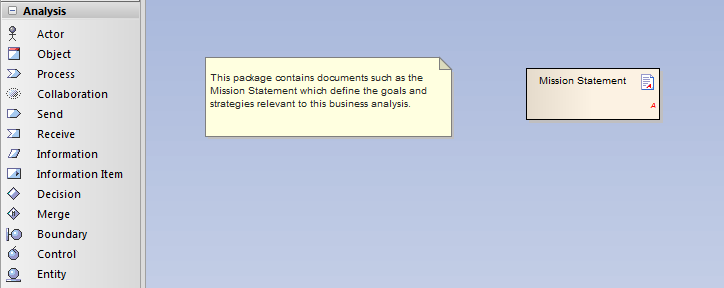


**Рис. 7.** Вид описания возможностей диаграммы  Business Process Model



**Рис. 8.** Ссылки на справочную информацию о диаграмме  Business Process Model

В рамках рассматриваемого проекта необходимо начать реализацию с постановки задачи, которая может быть определена в пакете Strategies, предназначенном для указания миссии, стратегии, бизнес-цели. Для описания пакета используется диаграмма Analysis, предложенная разработчиками Enterprise Architect и не входящая в стандарт UML (рис. 9)



**Рис. 9.** Диаграмма пакета Strategies

Диаграмма содержит текстовый документ, где предлагается описать постановку задачи. Для рассматриваемого проекта фрагмент постановки задачи имеет следующий вид:

**Постановка задачи**

Автоматизировать процесс управления информацией о клиентах, материалах, оборудовании, сотрудниках, услугах.

**Цель**

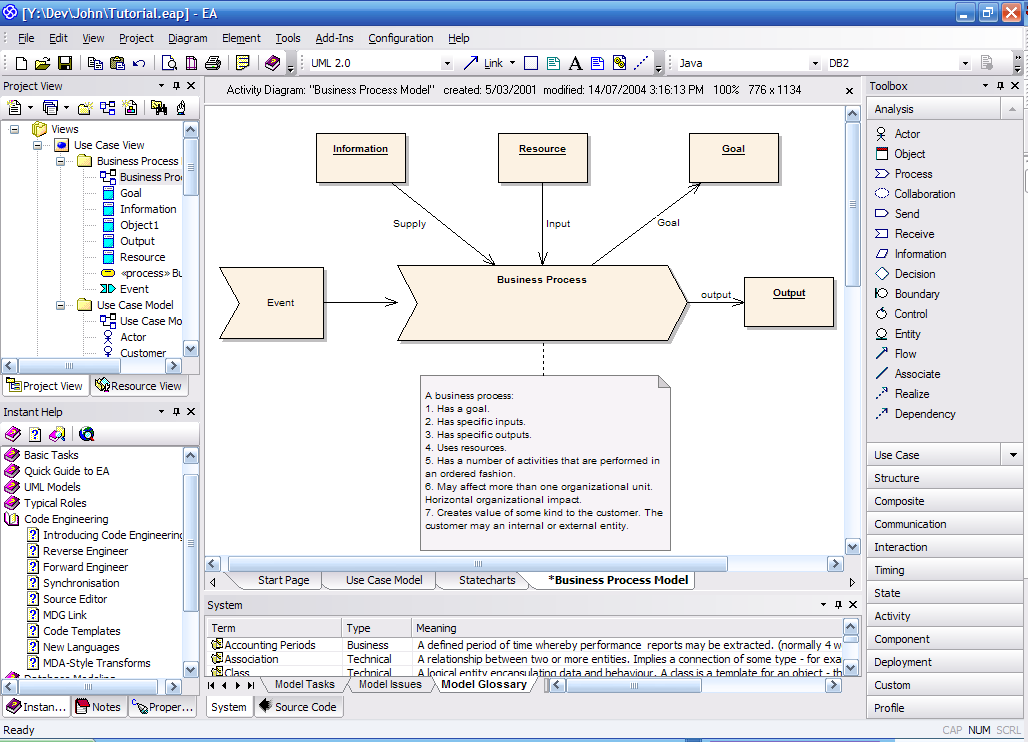
Обеспечить:

* Оперативность выполнения услуг,
* Учет и планирования использования материалов,
* Сбор информации о клиентах для повышения эффективности планирования оказания услуг,
* Учет состояния и оперативность ремонта оборудования.

**Бизнес процессы**

* Запись клиента на оказание услуг,
* Закупка материалов и оборудования,
* Оплата услуг,
* Отзывы,
* Ведение информации о клиентах,
* Ведение информации об оборудовании,
* Ведение информации о материалах,
* Ведение информации об услугах,
* Инвентаризация,
* Формирование скидок и др.

Затем необходимо детализировать бизнес процессы в виде моделей. Для этого предназначен пакет Business Workflows, где бизнес процессы описываются на диаграмме Analysis. Enterprise Architect предлагает для описания бизнес процессов модель Ericsson-Penker (рис. 10).



**Рис. 10.** Диаграмма модели бизнес процесса Ericsson-Penker

Элементы модели описаны в таблице (табл. 1).

**Таблица 1.** Элементы модели бизнес процесса Ericsson-Penker

|  |  |
| --- | --- |
| **Графический символ элемента модели** | **Описание** |
|  | Process. Наименование бизнес процесса. |
|  | Receive. Событие, инициирующее бизнес процесс. |
|  | Object типа Output. Информация, являющаяся результатом (выходом) бизнес процесса. |
|  | Object типа Goal. Цель бизнес процесса. Цель содержит описание выгод, которые принесет реализация бизнес процесса. |
|  | Object типа Information. Информация, необходимая для реализации бизнес процесса. |
|  | Object типа Resource. Ресурсы, необходимые для реализации бизнес процесса. |
|  | Отношение типа Associate |

Для примера приведем описание бизнес процесса «Запись клиента на оказание услуг» (рис. 11), как результата проектирования.



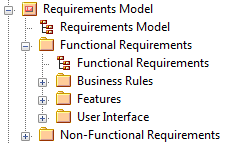
**Рис. 11.** Диаграмма бизнес процесса

Методология построения моделей бизнес процессов изучается в курсе «Моделирование бизнес процессов».

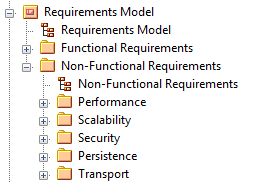
## Requirements Model

Пакет Requirements содержит:

* Пакет Functional Requirements содержит функциональные требования проекта (рис. 12).
  + **Пакет Business Rules** представляет собой каталог явных бизнес-правил, которые необходимы для реализации в рамках текущего проекта. Бизнес-правила обычно выполняются во время выполнения программы и управления обработкой информации и транзакциями.
  + **Пакет Features** описывает дискретные кусочки функционального поведения, которые дают конкретный результат.
  + **Пакет User Interface** пользователя содержит описания видимых экранов и форм конечного пользователя высокого уровня, которые необходимы для поддержки предлагаемой системы.
* Пакет Non-Functional Requirements содержит нефункциональные требования проекта (рис. 13), отвечающие за производительность, масштабируемость (расширяемость), безопасность, устойчивость и транспортируемость (передача данных).

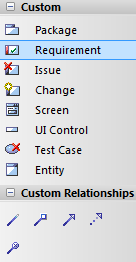
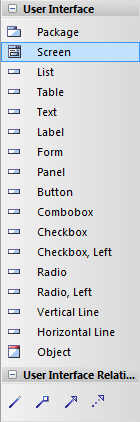


**Рис. 12.** Иерархия пакета Functional Requirements



**Рис. 13.** Иерархия пакета Non-Functional Requirements

В рамках проекта предлагается осуществить проектирование пакетов Business Rulesи User Interface, где описываются диаграммы Requirements и User Interface. Для построения диаграмм используется инструменты панели Custom и User Interface. Описание элементов диаграмм Requirements представлены в таблице (табл.2), элементы User Interface соответствуют элементам пользовательского интерфейса, используемых в современных информационных технологиях.

**Рис. 14.** Инструменты панели Custom и User Interface

**Таблица 2.** Элементы диаграммы Requirements

|  |  |
| --- | --- |
| **Графический символ элемента модели** | **Описание** |
|  | Requirement. Содержание требования. |
|  | Отношения типа:   * Ассоциация (Associate) * Агрегация (Aggregate) * Обобщение (наследование) (Generalize) * Реализация (Realize) * Зависимость (Dependency) |

Для примера приведем описание требований (рис. 15) и пользовательский интерфейс (рис. 16) проекта для бизнес процесса «Запись клиента на оказание услуг», как результата проектирования.

При проектировании интерфейса пользователя необходимо руководствоваться основными принципами эстетичности (однотипность элементов, выравнивание элементов, однотипность размеров элементов) и дружественности интерфейса (отсутствие перегруженности форм, подписи элементов, пользовательские подсказки и др.).

Формирование требований и пользовательского интерфейса осуществляется итеративно при активном взаимодействии с заказчиком. При обсуждении с заказчиком созданного интерфейса могут быть выявлены новые требования к проекту, которые необходимо отразить в модели.

****

**Рис. 15.** Модель Requirements

****

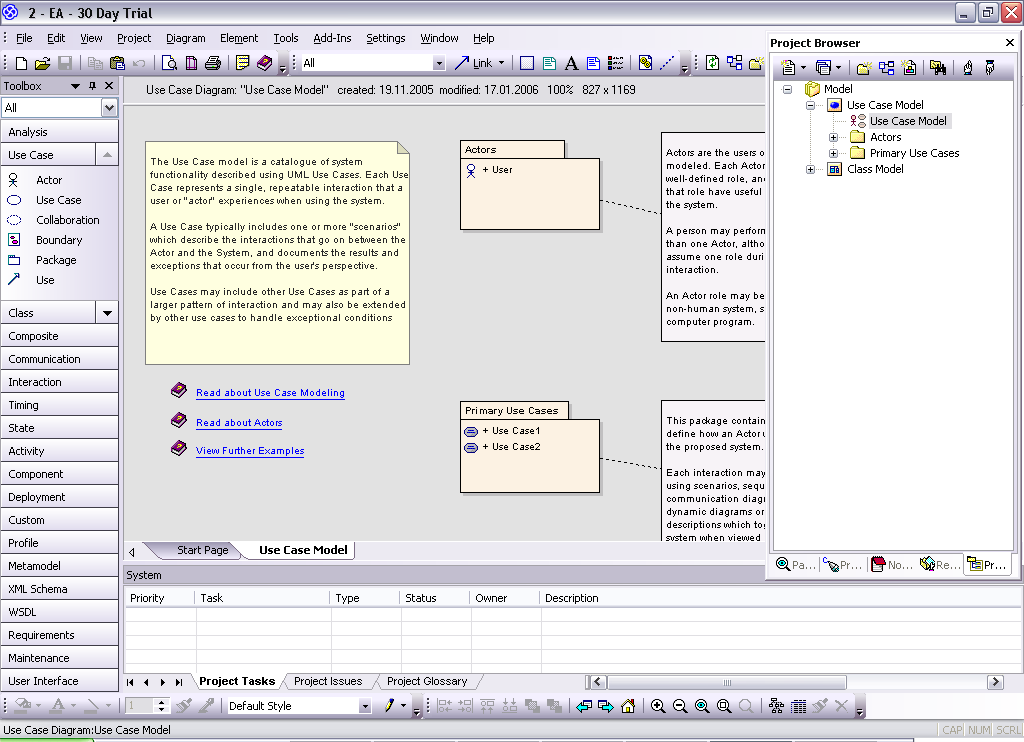
**Рис. 16.** Модель User Interface

## Use Case Model

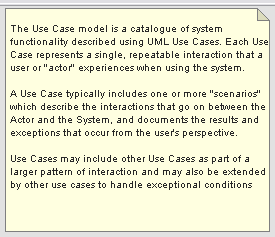
После завершения этапа формирования требований разработчик начинает описывать функции системы, для чего предназначена диаграмма Use Case.

Пакет Use Case Model содержит:

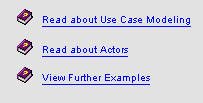
* Диаграмму  Use Case Model для краткого описания содержания основных пакетов диаграмм Actors и Primary Use Cases (рис. 17), где представлены эти пакеты с содержанием по умолчанию, описание возможностей диаграммы (рис. 18), комментарии к пакетам, ссылки на справочную информацию (рис. 19).
* Пакет Actors для хранения пакета диаграмм, описывающих актеров. По умолчанию создается Actor User. Actors обозначаются символом .
* Пакет Primary Use Cases для хранения пакета диаграмм, описывающих основные Use Case. По умолчанию создаются Use Cases: Use Case1 и Use Case2. Use Cases обозначаются символом .



**Рис. 17.** Окно проекта, отображающее диаграмму  Use Case Model

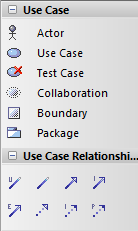


**Рис. 18.** Вид описания возможностей диаграммы Use Case Model



**Рис. 19.** Ссылки на справочную информацию по диаграмме  Use Case Model

Для построения диаграмм Use case используется инструменты панели Use Case (рис. 20). Описание элементов диаграммы Use case представлены в таблице (табл. 3).



**Рис. 20.** Инструменты панели Use Case

**Таблица 3.** Элементы диаграммы Use Case

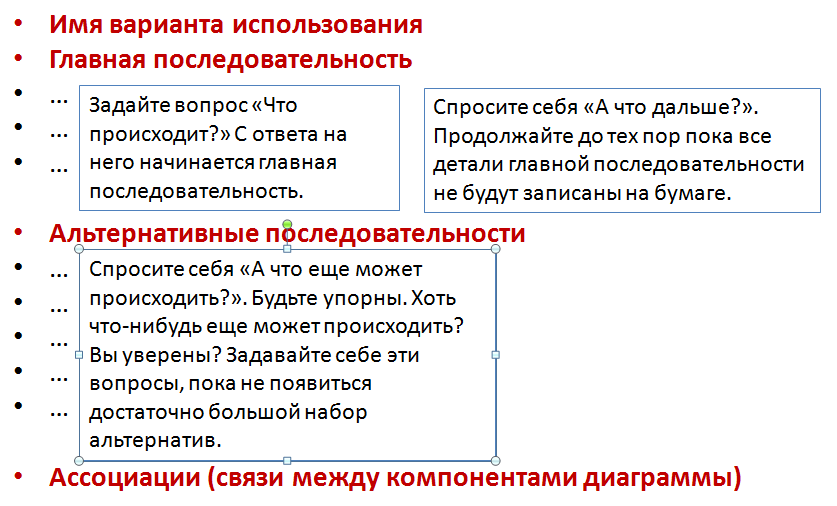
|  |  |
| --- | --- |
| **Графический символ элемента модели** | **Описание** |
|  | Use case. Содержание Use case (прецедента, вариант использования). Внутри эллипса содержится его краткое название или имя в форме глагола с пояснительными словами. |
|  | Actors. Любая внешняя по отношению к моделируемой системе сущность, которая взаимодействует с системой. |
|  | Отношения типа   * Ассоциации (association relationship); * Расширения (extend relationship). Один из вариантов использования может присоединять к своему поведению некоторое дополнительное поведение.      * Обобщения (generalization relationship); * Включения (include relationship). Заданное поведение для одного варианта использования включается в качестве составного компонента в последовательность поведения другого варианта использования. |

Для примера приведем описание диаграммы Use Case (рис. 21) для бизнес процесса «Запись клиента на оказание услуг», как результата проектирования.



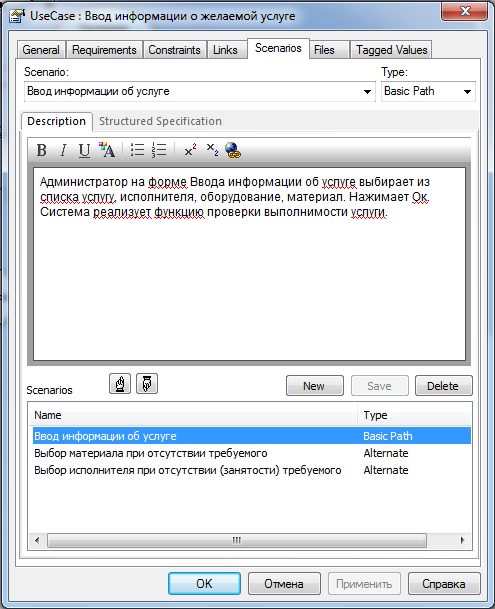
**Рис. 21.** Пример диаграммы Use Case

Каждый из Use case необходимо описать в виде сценария, где будут перечислены действия пользователя и ответные действия системы. Шаблон сценария представлен на рисунке (рис. 22).



**Рис. 22.** Шаблон сценария для описания Use Case (варианта использования)

Enterprise Architect позволяет описывать сценарии в спецификации Use case, на закладке Scenarios, с указанием главной и альтернативный последовательностей (рис. 23).



**Рис. 23.** Спецификация Use Case для описания сценариев

## Domain Model

Диаграмма предназначена для описания основных понятий предметной области. Построение модели предметной области начинается с выявления абстракций, существующих в реальном мире. Требования к программе меняются намного быстрее, чем реальный мир. Следовательно, создавая качественную модель реального мира, проектировщик обеспечивает стабильность проекта.

После выявления объектов предметной области необходимо установить отношения между ними. Основные отношения модели: обобщение и агрегация. Не нужно тратить время на выявление атрибутов и операций классов. Этот процесс будет выполняться позже при создании модели классов. Главная задача создать наиболее качественную модель, которую можно использовать в дальнейшем при проектировании новых систем в данной предметной области (т.е. переиспользование).

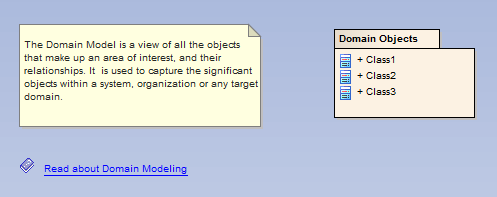
Лучшим источником классов является описание задачи и требований к системе. Необходимо начать выписывать предложения, характеризующие задачу, не забывая литературу предметной области, затем подчеркивать все существительные и именные группы. Имена существительные и именные группы становятся объектами и атрибутами. Глаголы и глагольные группы становятся операциями и ассоциациями. Родительный падеж показывает, что имя существительное должно быть атрибутом, а не объектом.

Далее следует отсеять из списка кандидатов на звание класса ненужные (избыточные или несущественные) и непригодные (слишком расплывчатые или понятия, лежащие за рамками модели) элементы. При построении модели можно принять предварительное решение об обобщениях (отношение наследования). Нужно искать предложения со словом «является». Также можно указать отношение агрегации (отношение вида «имеет»).

Данная диаграмма отсутствует в стандарте UML, однако EA предлагает средство реализации диаграммы.

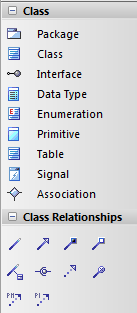
Пакет Domain Model содержит:

* Диаграмму  Domain Model для краткого описания содержания пакета диаграмм Domain Objects (рис. 24), где представлен этот пакет с содержанием по умолчанию, описание возможностей диаграммы, комментарии к пакетам, ссылки на справочную информацию.
* Пакет Domain Objects предназначен для хранения диаграммы Domain Objects и ее элементов, описывающих объекты (классы) предметной области. По умолчанию в данной папке создаются классы: Class 1, Class 2, Class 3.



**Рис. 24.** Содержание пакета Domain Model

Для построения диаграммы Domain Model используется инструменты панели Class (рис. 25). Описание основных элементов диаграммы Domain Model представлены в таблице (табл. 4).



**Рис. 25.** Инструменты панели Class

**Таблица 4.** Элементы диаграммы Domain Model

|  |  |
| --- | --- |
| **Графический символ элемента модели** | **Описание** |
|  | Class. Содержит имя класса. Имя класса указывается в верхней части прямоугольника. Имя формулируется как имя существительное, во множественном числе, пишется с заглавной буквы. |
|  | Отношения типа:   * Ассоциации (association relationship);      * Обобщения (generalization relationship); |

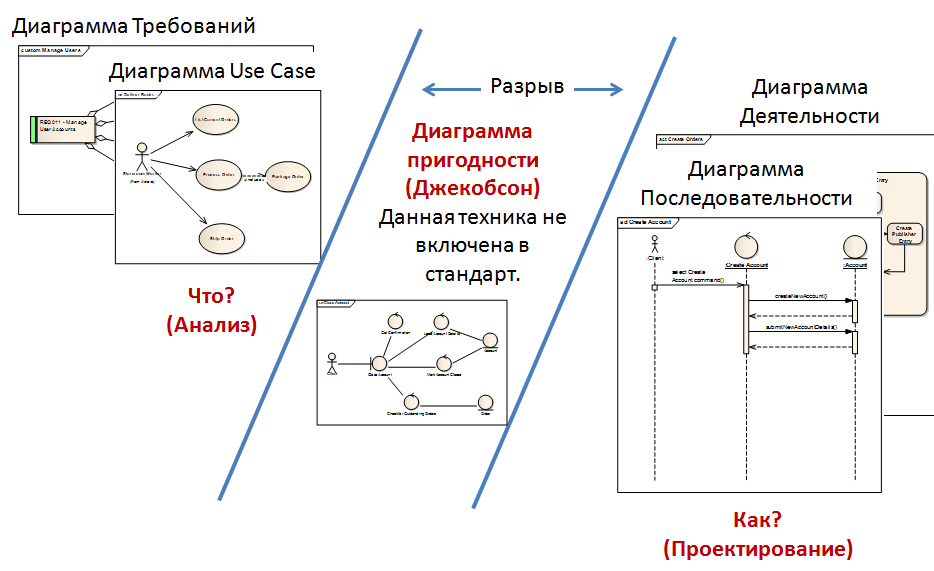
Для примера приведем описание диаграммы Domain Model (рис. 26) для бизнес процесса «Запись клиента на оказание услуг», как результата проектирования.



**Рис. 26.** Пример диаграммы Domain Model

## Диаграмма анализа пригодности

Следующим шагом проектирования в рамках рассматриваемого проекта является разработка диаграммы анализа пригодности, предложенной А. Джекобсоном. По его мнению, диаграммы UML условно можно разбить на две группы: «что?» и «как?». При переходе от анализа «что?» к этапу проектирования «как?» в методологии образуется «разрыв», который создает проблемы для проектировщиков. Для ликвидации данного разрыва и была предложена диаграмма анализа пригодности (рис. 27).



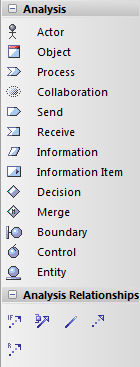
**Рис. 27.** «Разрыв» в понимании методологии UML

Диаграмма не входит в стандарт UML, однако EA предоставляется возможности для ее построения.

Для построения диаграммы используется инструменты панели Analysis (рис. 28). Описание основных элементов диаграммы анализа пригодности представлены в таблице (табл. 5).

На схемах (рис. 29, рис. 30) показано место диаграммы анализа пригодности в процессе проектирования.

В таблице (табл. 6) показаны правила создания отношений между элементами диаграммы.



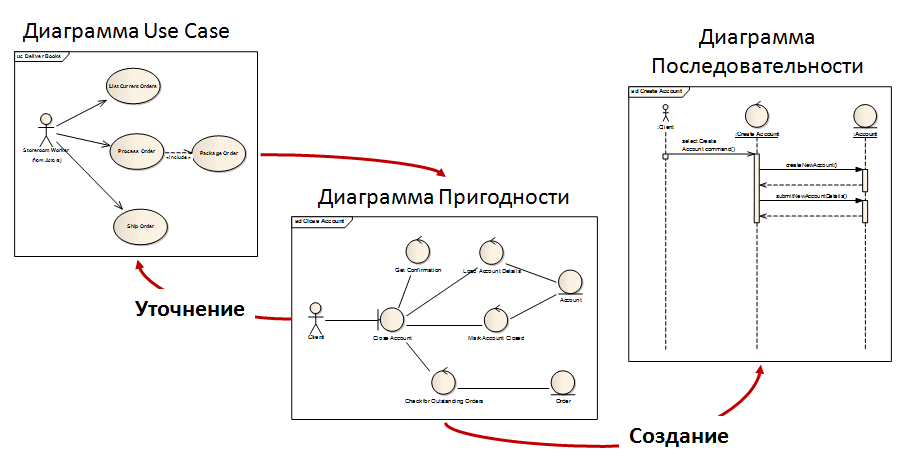
**Рис. 28.** Инструменты панели Analysis

**Таблица 5.** Элементы диаграммы анализа пригодности

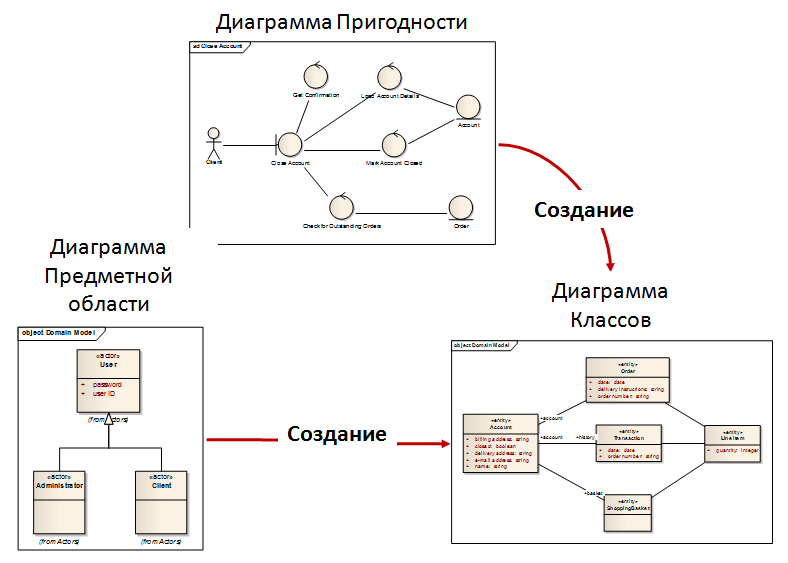
|  |  |
| --- | --- |
| **Графический символ элемента модели** | **Описание** |
|  | Boundary. Граничный объект – это объект, которым актер пользуется при взаимодействии с системой. |
|  | Entity. Сущностный объект – это обычно объект модели предметной области. |
|  | Control. Управляющий объект – (обычно называют контроллерами, т.к. им ничего не соответствует в реальном мире) это объект, выполняющий функцию «клея» между граничными и сущностными объектами. |
|  | Отношения типа:   * Ассоциации (association relationship).   На данном отношении указывается наименование действия, совершаемого пользователем или системой. |

**Таблица 6.** Правила образование связей между элементами диаграммы анализа пригодности

|  |  |
| --- | --- |
| **Разрешено** | **Запрещено** |
|  |  |



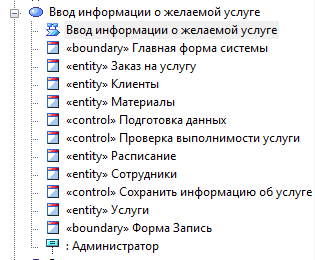
**Рис. 29.** Взаимодействия диаграмм анализа пригодности, Use case и Sequence



**Рис. 30.** Взаимодействия диаграмм анализа пригодности, Domain Model и Class

Опишем последовательность построения диаграммы анализа пригодности. Основой данной диаграммы является диаграмма Use case и диаграмма Domain Model. Необходимо на диаграмме Use case выделить основные Use case (прецеденты, варианты использования), провести анализ их сценариев, обнаружить все объекты (в том числе уже выделенные на диаграмме Domain Model), определить вид этих объектов: граничные, сущностные, управляющие, затем установить отношения между ними, соблюдая правила (табл. 6). В результате создания отношений могут быть выявлены новые объекты, не упоминаемые ранее при описании сценариев Use case.

В EA рекомендуется создавать диаграмму анализа пригодности для основных Use case, следовательно, диаграмма располагается в пакете Use Case Model, как подэлемент элемента Use Case (рис. 31). Диаграмма, т.к. использует инструмент Analysis, имеет следующее обозначение: .



**Рис. 31.** Пример расположения диаграммы анализа пригодности в проекте

Выявим объекты из сценария «Ввод информации об услуге» (рис. 23):

«Администратор на форме Ввода информации об услуге выбирает из списка услугу, исполнителя, оборудование, материал. Нажимает Ок. Система реализует функцию проверки выполнимости услуги».

Из текста можно выявить следующие объекты: граничный объект – форма Ввода информации об услуге; сущностные объекты – услуга, исполнитель (на диаграмме Domain Model данное понятие указано как Сотрудник), оборудование, материал; управляющий объект – функция проверки выполнимости услуги. Далее необходимо установить отношения между объектами. Исходя из текста, должны быть отношения между граничным и сущностными объектами, но согласно правилам установления отношений (табл. 6), между граничными и сущностными объектами отношений быть не может, а должны быть управляющие объекты, которые и обеспечивают взаимодействие указанных объектов. Таким образом, выявлен новый объект управляющего типа, который будет осуществлять подготовку данных для дальнейшего отображения их на форме.

Полное описание примера диаграммы анализа пригодности для бизнес процесса «Ввод информации о желаемой услуге» приведем на рисунке (рис. 32).



**Рис. 32.** Пример диаграммы анализа пригодности

## Sequence Model

В UML взаимодействие объектов рассматривается с точки зрения их коммуникации, т.е. обмена некоторой информации. Информация представляется в виде законченных сообщений. Следовательно, объекты отправляют и принимают сообщения друг друга. Для отображения последовательности отправки/приема сообщений с учетом времени используется диаграмма последовательности.

Диаграмма последовательностей (Sequence) – графическая модель, которая для определенного сценария варианта использования отображает генерируемые действующими лицами события и их порядок.

Основные элементы диаграммы: объекты и сообщения (рис. 33). Объекты располагаются в верхней части диаграммы. Не исключается ситуация, когда имя объекта может отсутствовать на диаграмме последовательности, а сам объект считается анонимным. Крайним слева изображается объект, который является инициатором взаимодействия. Правее изображается другой объект, который непосредственно взаимодействует с первым. Т.о. все объекты образуют некоторый порядок, определяемый их активностью.

Отличительная особенность диаграммы – наличие времени как самостоятельного измерения. Начальному моменту времени соответствует самая верхняя часть диаграммы.

Взаимодействие объектов реализуется посредством сообщений, которые посылаются одним объектами другими (рис. 33). Сообщения изображаются в виде горизонтальных стрелок с именем сообщения, и также образуют порядок по времени своего возникновения. Сообщения, расположенные выше, инициируются раньше, чем те, которые расположены ниже. Масштаб на диаграмме не указан, описывается только упорядоченность «раньше-позже».

Каждый объект имеет свою линию жизни (object lifeline) (рис. 33), которая изображается пунктирной вертикальной линией, ассоциированной с единственным объектом на диаграмме последовательности. Линия жизни служит для обозначения периода времени, в течение которого объект существует в системе. Если объект существует в системе постоянно, то и его линия жизни должна продолжаться по всей плоскости от самой верхней ее части до самой нижней.

Отдельные объекты, выполнив свою роль в системе, могут быть уничтожены (разрушены), чтобы освободить занимаемые ими ресурсы. Для таких объектов линия жизни обрывается в момент его уничтожения. Для обозначения момента уничтожения объекта в языке UML используется специальный символ в форме латинской буквы «X». Ниже этого символа пунктирная линия не изображается, поскольку соответствующего объекта в системе уже нет.



**Рис. 33.** Схема диаграммы последовательности (Sequence)

В процессе функционирования одни объекты могут находиться в активном состоянии, непосредственно выполняя определенные действия или в состоянии пассивного ожидания сообщений от других объектов. Чтобы явно выделить подобную активность объектов, в языке UML применяется специальное понятие, получившее название фокуса управления (focus of control) (рис. 33). Фокус управления изображается в форме вытянутого узкого прямоугольника, верхняя сторона которого обозначает начало получения фокуса управления объекта (начало активности), а ее нижняя сторона — окончание фокуса управления (окончание активности). Этот прямоугольник располагается ниже обозначения соответствующего объекта и может заменять его линию жизни, если на всем ее протяжении он является активным.

Каждое взаимодействие между объектами описывается совокупностью сообщений, которыми обмениваются объекты между собой. В этом смысле сообщение (message) представляет собой законченный фрагмент информации, который отправляется одним объектом другому. При этом прием сообщения означает:

* передача некоторой информации;
* инициирование выполнения определенных действий.

На диаграмме последовательности все сообщения упорядочены по времени своего возникновения. Каждое сообщение имеет направление от объекта, который инициирует и отправляет сообщение, к объекту, который его получает.

Наиболее распространенный вид сообщения используется для вызова процедур, выполнения операций или обозначения отдельных потоков управления. Начало этой стрелки всегда соприкасается с фокусом управления или линией жизни того объекта, который инициирует это сообщение. Конец стрелки соприкасается с линией жизни того объекта, который принимает это сообщение и выполняет в ответ определенные действия. При этом принимающий объект зачастую получает и фокус управления, становясь активным. Существуют стереотипы сообщений, названия которых говорят о себе: "call" (вызвать), "return" (возвратить), "create" (создать), "destroy" (уничтожить), "send" (послать).

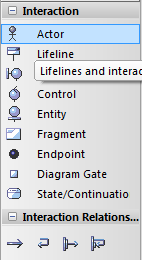
Сообщения могут иметь собственное обозначение операции, вызов которой они инициируют у принимающего объекта. В этом случае рядом со стрелкой записывается имя операции с круглыми скобками, в которых могут указываться параметры или аргументы соответствующей операции. Если параметры отсутствуют, то скобки все равно должны присутствовать после имени операции. Примерами таких операций могут служить следующие: "выдать клиенту наличными сумму (n)", "установить соединение между абонентами (а, Ь)", "сделать вводимый текст невидимым ()", "подать звуковой сигнал тревоги ()".

Если необходимо отобразить варианты сообщений в зависимости от логического условия, то рисуются две или более стрелки, выходящие из одной точки фокуса управления объекта. При этом соответствующие условия должны быть явно указаны рядом с каждой из стрелок в форме сторожевого условия. Условия должны взаимно исключать одновременную передачу альтернативных сообщений.

В отдельных случаях выполнение тех или иных действий на диаграмме последовательности может потребовать явной спецификации временных ограничений, накладываемых на сам интервал выполнения операций или передачу сообщений. В языке UML для записи временных ограничений используются фигурные скобки: {время\_ожидания\_ответа < 5 сек.} {время\_передачи\_пакета < 10 сек.}.

Диаграмма последовательности строится на основе диаграммы анализа пригодности. Все объекты, выделенные на этапе создания диаграммы анализа пригодности, помещаются на диаграмму последовательности, затем выявленные отношения между объектами преобразуются в сообщения между объектами, которые на данной диаграмме уточняются вплоть до имен операций.

Для построения диаграммы используется инструменты панели Interaction (рис. 34).



**Рис. 34.** Инструменты панели Interaction

Пример диаграммы последовательности представлен на рисунке (рис. 35).



**Рис. 35.** Фрагмент примера диаграммы последовательности (Sequence)

В методологии использования UML рекомендуется создавать диаграммы последовательности для описания основных или сложных для понимания процессов функционирования системы, т.е. для каждого из основных Use case создается соответствующая диаграмма Sequence. Следовательно, диаграмма Sequence располагается в пакете Use Case Model обозначается символом .

## Class Model

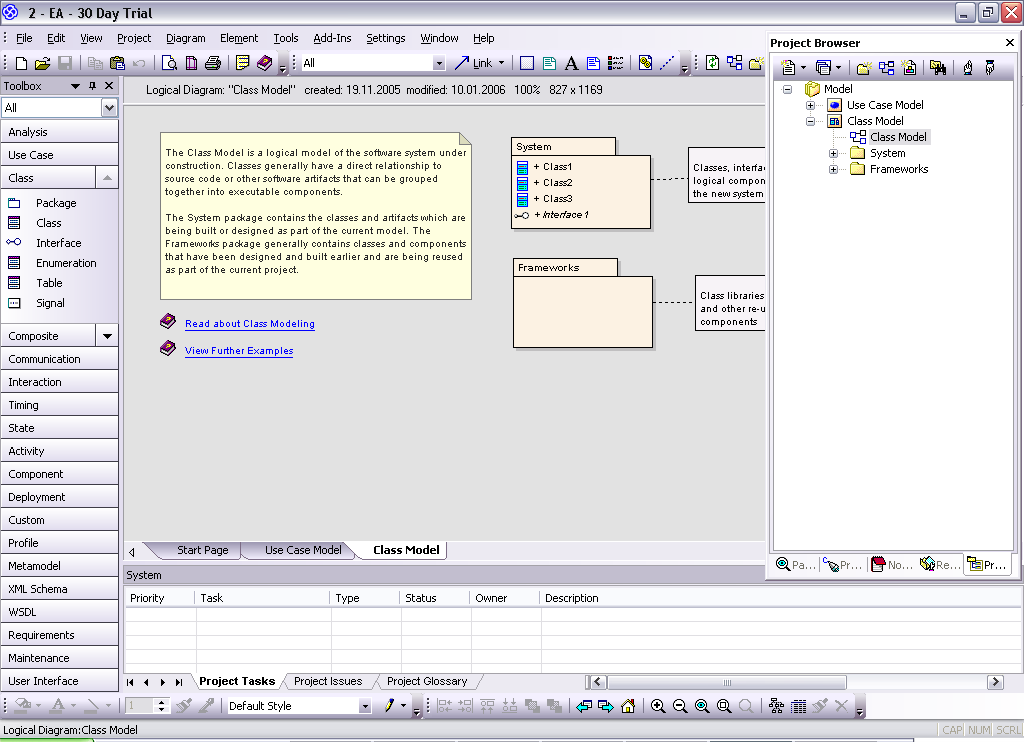
Следующим этапом создания проекта может быть создание диаграммы классов (Class Model).

Диаграммы классов – графическое изображение концептуальной модели предметной области, т.е. основных сущностей (понятий) и их взаимосвязей. Диаграмма классов служит для представления статической структуры модели системы, которая не зависит от времени. Основные элементы диаграмм классов: класс, атрибут, отношение.

Диаграмма классов может также содержать интерфейсы, пакеты, отношения и даже отдельные экземпляры, такие как объекты и связи.

Пакет Class Model содержит:

* Диаграмму  Class Model для краткого описания содержания основных пакетов диаграмм System и Frameworks (рис. 36), где представлены эти пакеты с содержанием по умолчанию, описание возможностей диаграммы, комментарии к пакетам, ссылки на справочную информацию.
* Пакет System для хранения пакета диаграмм, описывающих классы информационной системы. По умолчанию в данной папке создаются классы: Class 1, Class 2, Class 3 и Interface 1 (этот класс в данной работе не рассматривается).
* Пакет Frameworks для хранения пакета диаграмм, описывающих внешние компоненты и классы проекта (в данной работе не рассматривается).



**Рис. 36.** Окно проекта, отображающее диаграмму  Class Model

Элементы диаграммы классов представлены в таблице (табл. 7). Возможные отношения между классами рассмотрены в таблице (табл. 2).

**Таблица 7.** Элементы диаграммы классов (Class Model)

|  |  |
| --- | --- |
| **Графический символ элемента модели** | **Описание** |
|  | Class. Имя класса должно быть уникальным в пределах пакета. Оно указывается в первой верхней секции прямоугольника.  В дополнение к общему правилу наименования элементов языка UML, имя класса записывается по центру секции имени полужирным шрифтом и должно начинаться с заглавной буквы. Рекомендуется в качестве имен классов использовать существительные, записанные по практическим соображениям без пробелов. |
|  | Abstract class. Класс может не иметь экземпляров или объектов. В этом случае он называется **абстрактным классом**, а для обозначения его имени используется наклонный шрифт (курсив) или перед именем класса указывают стереотип «abstract». |
|  | Interface. Интерфейсом в UML называют класс, содержащий только объявление операций. |
|  | Table. Класс, являющийся элементом модели данных, который в дальнейшем в системе будет представлен в виде реляционной таблицы данных. |
|  | Object. Объект (object) является отдельным экземпляром класса, который создается на этапе выполнения программы. Он имеет свое собственное **имя** и **конкретные значения атрибутов**. В общем случае запись имени объекта представляет собой строку текста **"имя объекта:имя класса",** разделенную двоеточием. |

Диаграмма классов создается итеративно по мере получения новых данных о предметной области, функциях системы и алгоритмах их реализации. Таким образом, диаграмма классов создается на основе диаграмм предметной области (Domain Model), анализа пригодности, диаграммы последовательности (Sequence), а также, при необходимости, всех уже созданных диаграмм в проекте. Все классы, выделенные на указанных диаграммах, должны быть представлены на диаграмме классов, где детализируется информация о структуре и поведении классов, т.е. детально описываются атрибуты и операции классов. Напомним, что атрибуты выявляются, в том числе, на диаграммах требований (Requirements) и вариантов использования (Use case), операции детально перечислены на диаграмме последовательности (Sequence).

Отношения между классами определены в процессе создания диаграмм предметной области (Domain Model), анализа пригодности и диаграммы последовательности (Sequence). На диаграмме классов отношения типа ассоциации должны быть описаны более детально, как минимум определена их мощность (Multiplicity).

Уточним возможности описания характеристик классов, атрибутов и операций (рис. 37).

Описание (спецификация) атрибута (рис. 38) может, помимо имени, включать: тип, описание видимости и значение по умолчанию. Для этого используют следующий формат:

***<признак видимости> <имя>[кратность]:<тип> = <значение по умолчанию>,***

где **признак (квантор) видимости** может принимать одно из трех значений:

* + «**+**» - общий или общедоступный (**public**), атрибут с этой областью видимости доступен или виден из любого другого класса пакета, в котором определена диаграмма;
  + «**#**» - защищенный (**protected**), атрибут с этой областью видимости недоступен или невиден для всех классов, за исключением подклассов данного класса;
  + «**-**» - скрытый (**private**), атрибут с этой областью видимости недоступен или невиден для всех классов без исключения.
* **Имя атрибута** представляет собой строку текста, которая используется в качестве идентификатора соответствующего атрибута и поэтому должна быть уникальной в пределах данного класса. Имя атрибута является единственным обязательным элементом синтаксического обозначения атрибута.
* **Кратность атрибута** характеризует общее количество конкретных атрибутов данного типа, входящих в состав отдельного класса. В общем случае кратность записывается в форме строки текста в квадратных скобках после имени соответствующего атрибута:

***[нижняя\_граница1 .. верхняя\_граница1, нижняя\_граница2.. верхняя\_граница2, ..., нuжняя\_гpaнuцa k .. верхняя\_граница k]***

[1.:\*] означает, что кратность атрибута может принимать любое положительное целое значение большее или равное 1.

[1..5] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 4, 5.



**Рис. 37.** Спецификация класса. Вкладка Detail

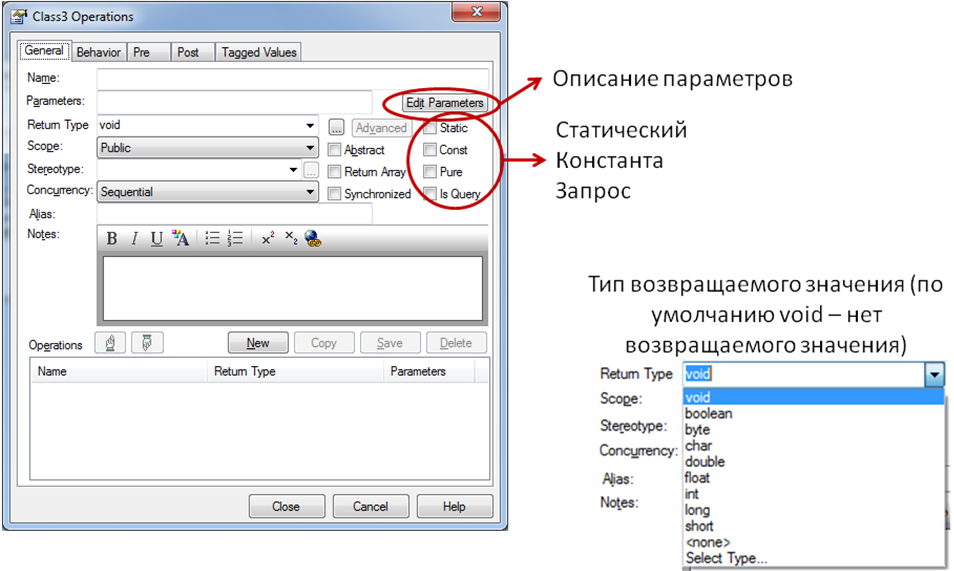


**Рис. 38.** Спецификация класса. Спецификация атрибутов

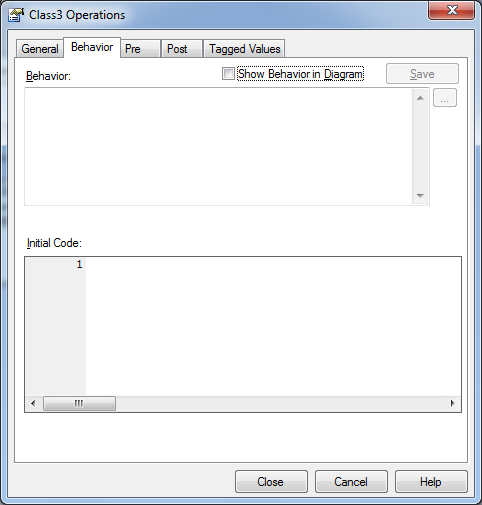
Описание (спецификация) операций класса имеет следующий вид (рис. 39-40):

***<признак видимости> <имя>(<список параметров>):******<тип возвращаемого значения>.***

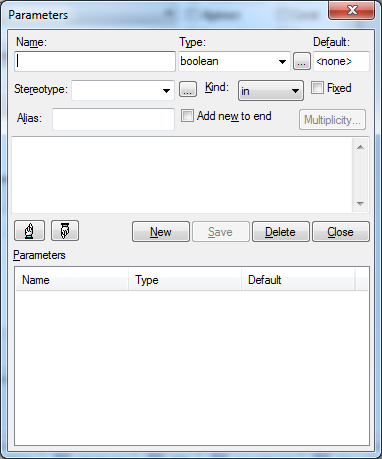
* **Признак видимости** принимает значения, аналогичные значениям признака видимости для атрибутов.
* **Имя** представляет собой строку текста, которая используется в качестве идентификатора соответствующей операции и поэтому должна быть уникальной в пределах данного класса.
* Имя операции является единственным обязательным элементом синтаксического обозначения операции.
* **Вид параметра** — одно из ключевых слов «**in**», «**out**» или «**inout**» со значением «in» по умолчанию, в случае, если вид параметра не указывается.
* **Имя параметра** - идентификатор соответствующего формального параметра.
* **Выражение типа** является зависимой от конкретного языка программирования спецификацией типа возвращаемого значения для соответствующего формального параметра.
* **Значение параметра по умолчанию** - выражение для значения формального параметра, синтаксис которого зависит от конкретного языка программирования и подчиняется принятым в нем ограничениям.



**Рис. 39.** Спецификация класса. Спецификация операций



**Рис. 40.** Спецификация класса. Спецификация операций. Возможность написания кода операции



**Рис. 41.** Спецификация класса. Параметры операций

Пример диаграммы классов для бизнес процесса «Ввод информации о желаемой услуге» приведем на рисунке (рис. 42).



**Рис. 42.** Пример диаграммы Class Model

## Activity Model

Диагра́мма де́ятельности (Activity)  — диаграмма, на которой показано разложение некоторой деятельности на её составные части. Под деятельностью (*activity*) понимается спецификация (описание) исполняемого поведения в виде последовательного и параллельного выполнения подчинённых элементов — вложенных видов деятельности и отдельных действий, соединённых между собой потоками, которые идут от выходов одного узла ко входам другого.

Описание основных элементов диаграммы деятельности представлены в таблице (табл. 8). Для построения диаграммы используется инструменты панели Activity (рис. 43).

**Таблица 8.** Элементы модели Activity

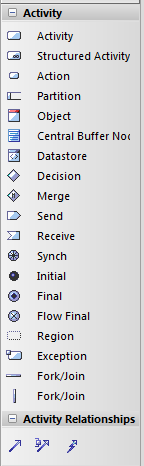
|  |  |
| --- | --- |
| **Графический символ элемента модели** | **Описание** |
|  | Initial. Символ начала описания некоторого процесса (деятельности). |
|  | Activity. Символ для описания содержания деятельности. |
|  | Decision. Символ принятия решения. Обеспечение возможности разветвления процесса по заданному условию. |
|  | Fork/Join. Символ объединения или разделения параллельных процессов (линии синхронизации). |
|  | Partition (дорожка). Символ, определяющий область ответственности описываемого элемента проектирования. |
|  | Final. Символ завершения процесса (деятельности). |

В зависимости от степени детализации диаграммы деятельностей так же, как диаграммы классов, используют на разных этапах разработки.

На этапе анализа требований и уточнения спецификаций диаграммы деятельностей *позволяют конкретизировать основные функции разрабатываемого программного обеспечения* (рис. 44). На примере диаграммы описывается взаимодействие Администратора и разрабатываемой Системы. Необходимо отметить, что данная диаграмма располагается в проекте в зависимости от назначения. Т.к. диаграмма описывает алгоритм функционирования разрабатываемой системы, то, следовательно, она располагается на уровне описания функций, т.е. в пакете Use case (рис. 45).

Не менее важная область применения диаграммы Activity связана с моделированием бизнес-процессов. В этом случае диаграмму необходимо располагать в пакете Business Process Model.

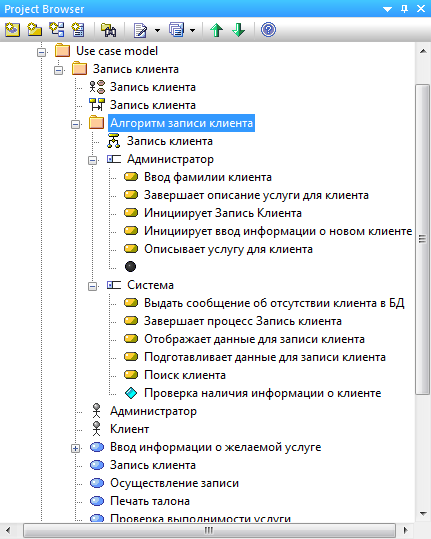
Диаграммы деятельности могут быть использованы для *спецификации алгоритмов* вычислений или потоков управления в программных системах. В этом случае диаграмму необходимо располагать в пакете Class Model, как подэлемент класса, алгоритм которого описывается.



**Рис. 43.** Инструменты панели Activity



**Рис.44.** Пример диаграммы Activity, описывающей взаимодействие Администратора и Системы при реализации бизнес процесса «Запись клиента»



**Рис. 45.** Расположение диаграммы Activity в проекте

## State Model

Теоретической основой диаграммы State (диаграммы состояний) является понятие «автомат». «Автомат» описывает поведение объекта в форме последовательности состояний, которые охватывают все этапы его жизненного цикла, начиная от создания объекта и заканчивая его уничтожением.

Для описания «автомата» используются понятия: состояние и переход. Отличие между состоянием и переходом: длительность нахождения системы в некотором состоянии значительно превышает время, которое затрачивается на переход из одного состояния в другое. Предполагается, что переход осуществляется мгновенно.

Представим состояния любого технического устройства (телевизор, телефон и др.) на самом общем уровне. Тогда вводятся два состояния «исправен», «неисправен», и два перехода – «выход из строя» и «ремонт», что отображает диаграмма:



Моделирование на основе автоматной модели обладает рядом условий:

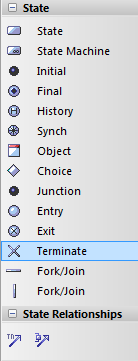
* Автомат не запоминает историю перемещения из состояния в состояние.
* В каждый момент времени автомат может находиться в одном и только в одном состоянии.
* Время не включается в понятие автомата, т.е. длительность нахождения в состоянии или достижения некоторого состояния не указывается.
* Количество состояний автомата должно быть конечным.
* Автомат не содержит изолированных состояний, т.е. для каждого состояния (кроме начального) должно быть определено предыдущее состояния, каждый переход должен соединять два состояния. Допускается переход из состояния в себя («петля»).
* Автомат не должен содержать конфликтных переходов (т.е. переход в два и более состояний, без указания условий перехода).

Диаграммы состояний – диаграмма, описывающая возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели программной системы в течении его жизненного цикла. Диаграмма состояний служит для представления динамической составляющей модели системы.

Основные элементы диаграмм состояний: состояние, переход, событие. Описание основных элементов диаграммы состояний представлены в таблице (табл. 9). Для построения диаграммы используется инструменты панели State (рис. 46).

**Таблица 9.** Основные элементы модели State

|  |  |
| --- | --- |
| **Графический символ элемента модели** | **Описание** |
|  | Initial. Символ момента начала описания некоторого поведения. |
|  | State. Символ состояния. Рекомендуется в качестве имени использовать глаголы в настоящем времени (звенит, печатает, ожидает) или причастия (занято, передано, получено). |
|  | StateMachine. Символ состояния, которое имеет подсостояния. |
|  | Choice. Символ ветвления. |
|  | Final. Символ момента окончания описания некоторого поведения. |
|  | **Простой** переход (simple transition) представляет собой отношение между двумя последовательными состояниями, которое указывает на факт смены одного состояния другим. На переходе указывается **имя события**.  Переход осуществляется при наступлении некоторого **события**:   * + Окончания выполнения действий (метка do);   + Получения объектом сообщения;   + Приема сигнала и т.д.   или выполнения некоторого **сторожевого условия**, т.е. условие принимает значение «истина». |



**Рис. 46.** Инструменты панели State

Диаграмма состояний представляет собой ориентированный граф, вершины которого соответствуют состояниям, а дуги - переходам. Поведение моделируется как последовательное перемещение по графу состояний от вершины к вершине по связывающим их дугам с учетом их ориентации.

Состояние может иметь так называемые метки:

***<метка-действия '/' выражение-действия>.***

**Стандартные метки:**

* **entry** (англ. – вход) – действие, которое выполняется в момент входа в данное состояние (входное действие); Например, создать соединение с базой данных **entry / createConnect()**;
* **exit** (англ. – выход) – действие, которое выполняется в момент выхода из данного состояния (выходное действие); Например, закрыть соединение с базой данных **exit / closeConnect()**;
* **do** (англ. – выполнять) – выполняющаяся деятельность ("do activity") в течение всего времени, пока объект находится в данном состоянии. Находясь в состоянии, объект может бездействовать и ждать наступления некоторого события, а может выполнять длительную операцию. Например, рассчитать допускаемые скорости **do / calculateVdop()**. Допускается указывать несколько операций в виде отдельных строк, каждая из которых начинается с метки **do**, или в виде одной строки, операции в которой отделены друг от друга точкой с запятой;
* **defer** (англ. – отложить) - событие, обработка которого предписывается в другом состоянии, но после того, как все операции в текущем будут завершены. Например, отображение на экране сообщения об ошибках в исходных данных **defer / showDataError()**;
* **newTarget** (англ. – новое задание) – внутренний переход, предписывающий обработку новых событий, не покидая текущего состояния. При выполнении внутреннего перехода повторно не выполняются действия при входе или выходе из состояния. Например, временная остановка (прерывание) расчета допускаемых скоростей, **newTarget / pauseCalculateVdop()**.

Переход между состояниями может быть помечен строкой текста, которая имеет следующий общий формат:

***<сигнатура события>'['<сторожевое условие>']' /<выражение действия>****.*

При этом сигнатура события описывает некоторое событие с необходимыми аргументами:

***<сигнатура события> это:   
<имя события>'('<список параметров, разделенных запятыми>')'.***

На рисунке (рис. 47) представлен пример диаграммы состояний, описывающей поведение класса Проверка выполнимости услуги для бизнес процесса Запись на выполнение услуги.

****

**Рис.47.** Пример диаграммы состояний (State) для класса Проверка выполнимости услуги

Диаграмма состояний описывает состояния конкретного класса, поэтому диаграмма располагается в пакете Class Model, как подэлемент конкретного класса, и обозначается символом .

## Component Model

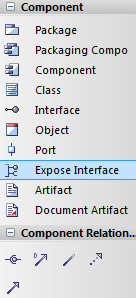
Следующие рассматриваемые диаграммы, диаграммы компонентов (Component) и развертывания (Deployment), относятся к диаграммам реализации (implementation diagrams) создаваемой программной системы.

Диаграмма компонентов описывает особенности физического представления системы, определяет архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный  и исполняемый код.

Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними (табл. 10). Для построения диаграммы используется инструменты панели Component (рис. 48).

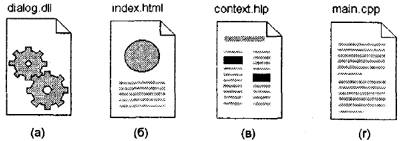
**Таблица 10.** Элементы модели Component

|  |  |
| --- | --- |
| **Графический символ элемента модели** | **Описание** |
|  | Component. Символ описания компонента. |
|  | Символ описания компонента, имеющего вложение. |



**Рис. 48.** Инструменты панели Component

Поскольку компонент представляет отдельный модуль кода, иногда его комментируют с указанием дополнительных графических символов, иллюстрирующих конкретные особенности его реализации.



Эти дополнительные обозначения для примечаний не специфицированы в языке UML. Однако их применение упрощает понимание диаграммы компонентов, существенно повышая наглядность физического представления.

В языке UML выделяют три вида компонентов:

* компоненты развертывания, которые обеспечивают непосредственное выполнение системой своих функций: динамически библиотеки с расширением dll , Web-страницы с расширением html и файлы справки с расширением hlр .
* компоненты-рабочие продукты: файлы с исходными текстами программ;
* компоненты исполнения: файлы с расширением ехе.

Другой способ спецификации различных видов компонентов — явное указание стереотипа компонента перед его именем. В языке UML для компонентов определены следующие стереотипы:

* **Библиотека** (library) — представляется в форме динамической или статической библиотеки.
* **Таблица** (table) — представляется в форме таблицы базы данных.
* **Файл** (file) — представляется в виде файлов с исходными текстами программ.
* **Документ** (document) — представляется в форме документа.
* **Исполнимый** (executable) — компонент, который может исполняться в узле.

Пример диаграммы компонентов приведем на рисунке (рис. 49).



**Рис. 49.** Пример диаграммы компонентов

## Deployment Model

Если разрабатывается простая программа, которая может выполняться локально на компьютере пользователя, не задействуя никаких периферийных устройств и ресурсов, то в этом случае нет необходимости в разработке дополнительных диаграмм.

Однако при разработке корпоративных приложений необходимость дополнительного аппаратного обеспечения очевидна (рис. 50).



**Рис. 50.** О необходимости диаграммы развертывания

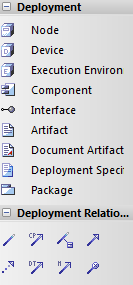
Диагра́мма развёртывания (Deployment diagram) моделирует физическое развертывание компонентов на узлах.

Например, чтобы описать веб-сайт, диаграмма развертывания должна показывать, какие аппаратные компоненты («узлы») существуют (например, веб-сервер, сервер базы данных, сервер приложения), какие программные компоненты работают на каждом узле (например, веб-приложение, база данных), и как различные части этого комплекса соединяются друг с другом.

Описание основных элементов диаграммы развертывания представлены в таблице (табл. 11). Для построения диаграммы используется инструменты панели Deployment (рис. 51).

**Таблица 11.** Элементы модели Deployment

|  |  |
| --- | --- |
| **Графический символ элемента модели** | **Описание** |
|  | Node. **Узел** представляет собой некоторый физически существующий элемент системы, обладающий некоторым вычислительным ресурсом:   * + Наличие некоторого объема электронной или магнитооптической памяти   + и/или процессора,   + другие механические или электронные устройства, такие как датчики, принтеры, модемы, цифровые камеры, сканеры и манипуляторы. |
|  | Node. Device – некоторое устройство. В качестве дополнения к имени узла могут использоваться различные стереотипы (хотя в языке UML стереотипы для узлов не определены):   * + "процессор", "датчик", "модем", "сеть", "консоль" и др., которые самостоятельно могут быть определены разработчиком;   + на диаграммах развертывания допускаются специальные обозначения для различных физических устройств. |
|  | Node. PC – Персональный компьютер |
|  | Node. Server – сервер. |
|  | Node. Store – хранилище информации. |



**Рис. 51.** Инструменты панели Deployment

Пример диаграммы развертывания приведем на рисунке (рис. 52).

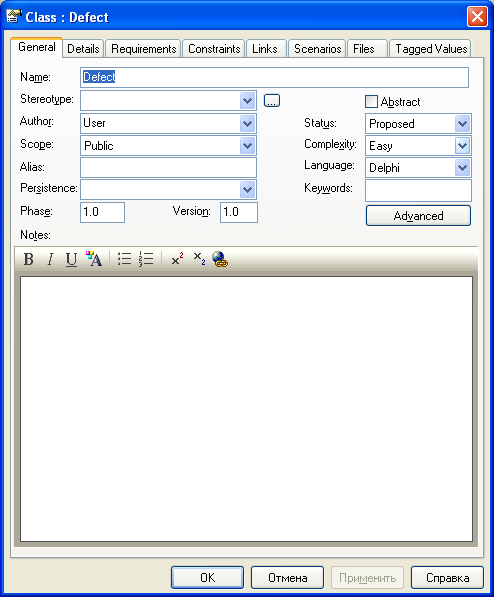


**Рис. 52.** Пример диаграммы Deployment

# Программная реализация

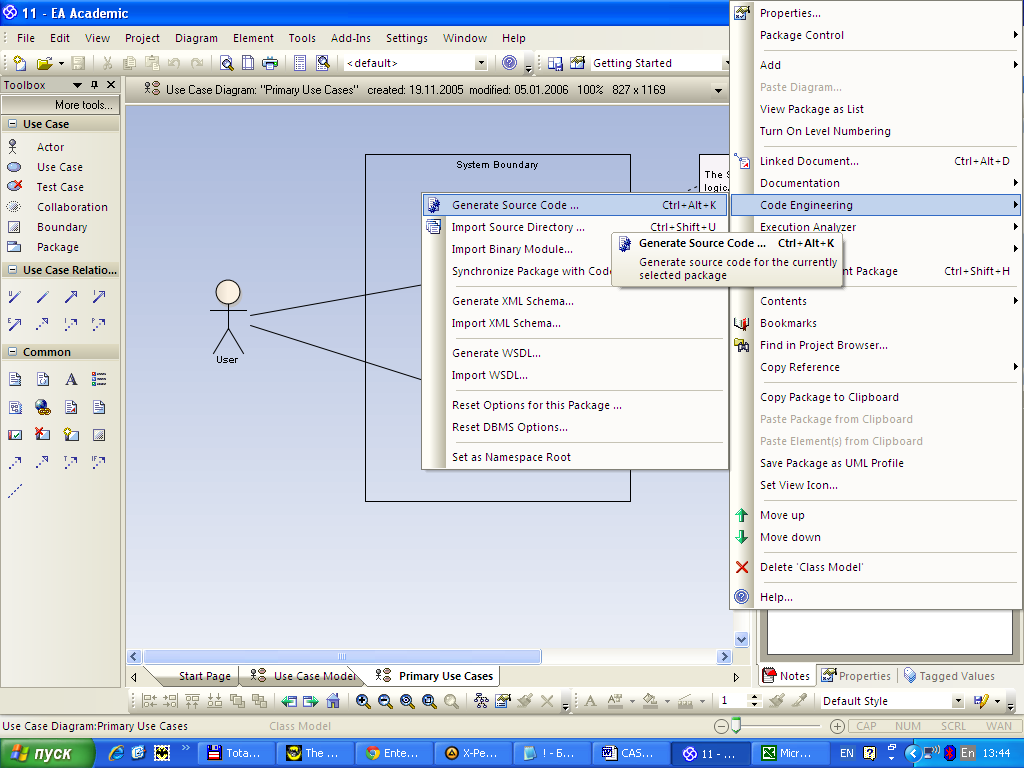
Enterprise Architect предоставляет для автоматической генерации программного кода широкий выбор технологических платформ. Программный код имеет «скелетный» вид, затем этот код дополняется необходимыми элементами в целевой среде программирования. При этом возможно обратное преобразование программного кода в модель на языке UML.

Рассмотрим пример генерации кода для языка Object Pascal (Delphi). Чтобы сгенерировать код на основе диаграммы классов необходимо для каждого класса в его свойствах определить целевой язык программирования (Рис.53).

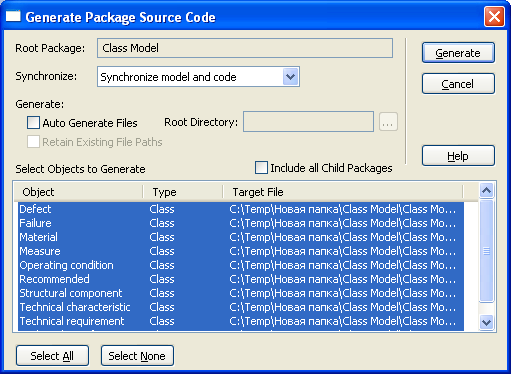


**Рис. 53.** Свойства класса, указание целевого языка программирования (Delphi)

Чтобы сгенерировать программный код необходимо выделить модель классов в Project Browser, вызвать контекстное меню и выбрать пункт Code Engineering (Рис.54) и определить настройки генератора кода (Рис.55).



**Рис. 54.** Вызов генератора программного кода



**Рис. 55.** Генератор программного кода

В результате генерации программного кода будет получен следующий «скелетный» код:

{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}

{\*

{\* uDefect.pas

{\* Delphi Implementation of the Class Defect

{\* Generated by Enterprise Architect

{\* Created on: 11-мар-2015 13:37:23

{\* Original author: User

{\*

{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}

unit uDefect;

interface

type

Defect = class

public

Name: string;

Value: string;

constructor Create; overload;

destructor Destroy; override;

end;

implementation

{implementation of Defect}

constructor Defect.Create;

begin

inherited Create;

end;

destructor Defect.Destroy;

begin

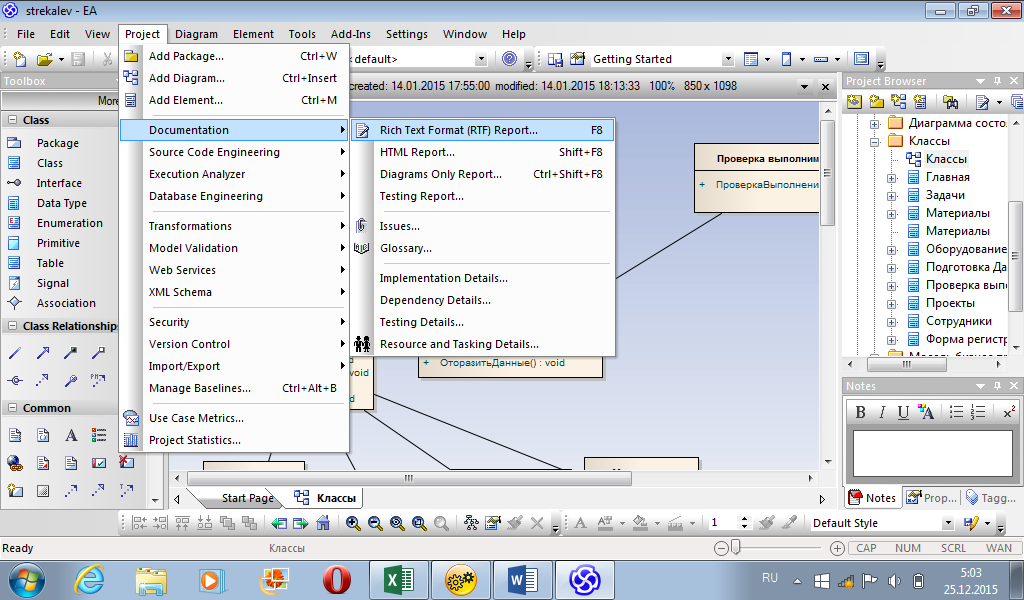
inherited Destroy;

end;

end.

# Формирование отчетов[[1]](#footnote-2)

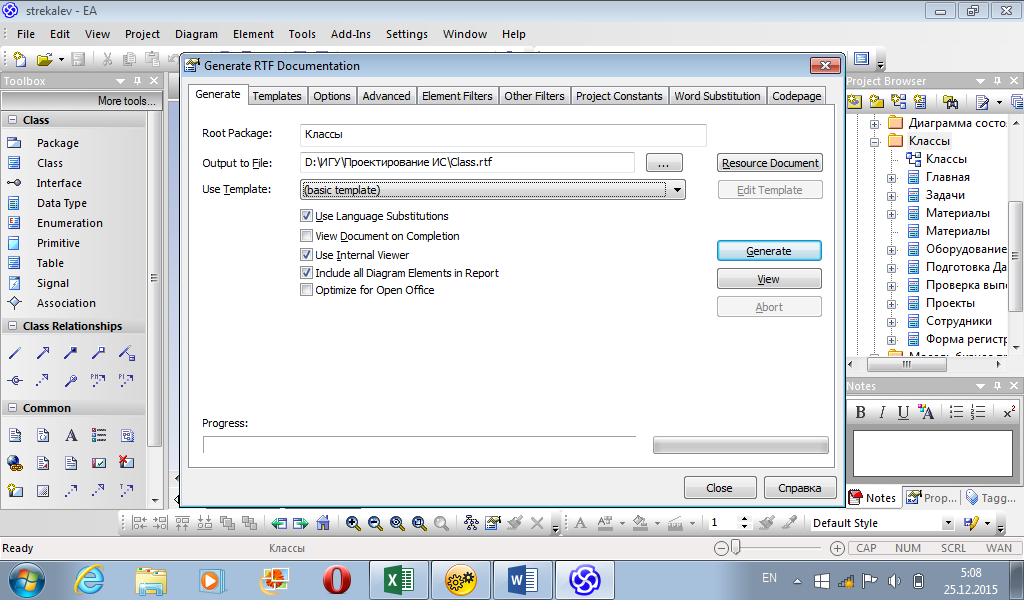
Для формирования отчетов в EA в пункте меню Project выбираем пункт Documentation, в котором указываем Rich Text Format (RTF) Report (рис. 56).



**Рис. 56.** Выбор меню для создания отчета

Затем на вкладке Generate (рис. 57) в полях:

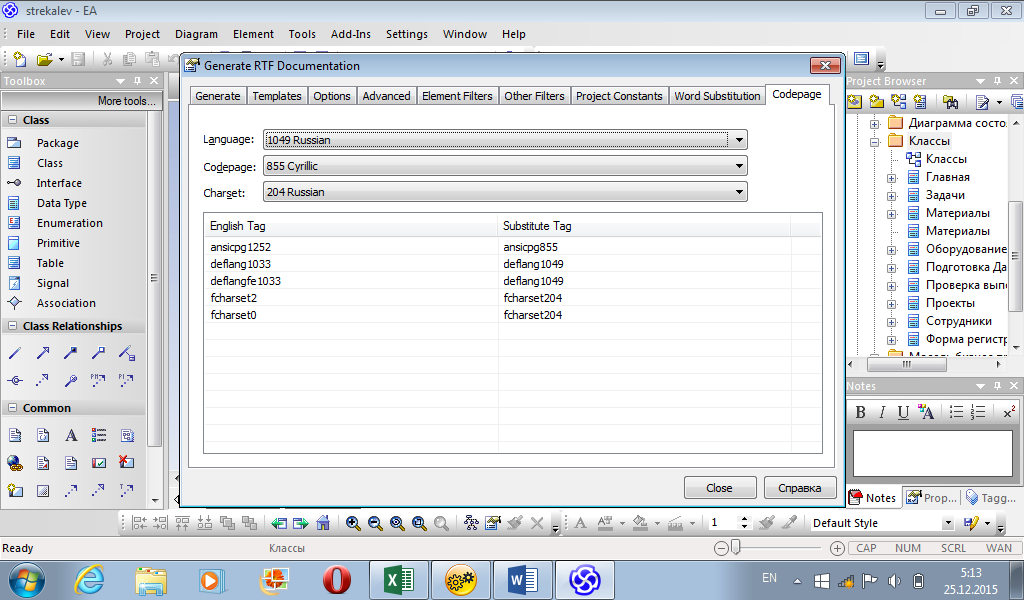
* Root Package – указываем диаграмму, для которой будет создаваться отчет, по умолчанию указана диаграмма, которая активна в окне Project Browser;
* Output To File – указываем имя файла создаваемого отчета;
* Template – выбираем тип используемого шаблона (рекомендуется для всех отчетов использовать базовый шаблон).



**Рис. 57.** Параметры создания отчета

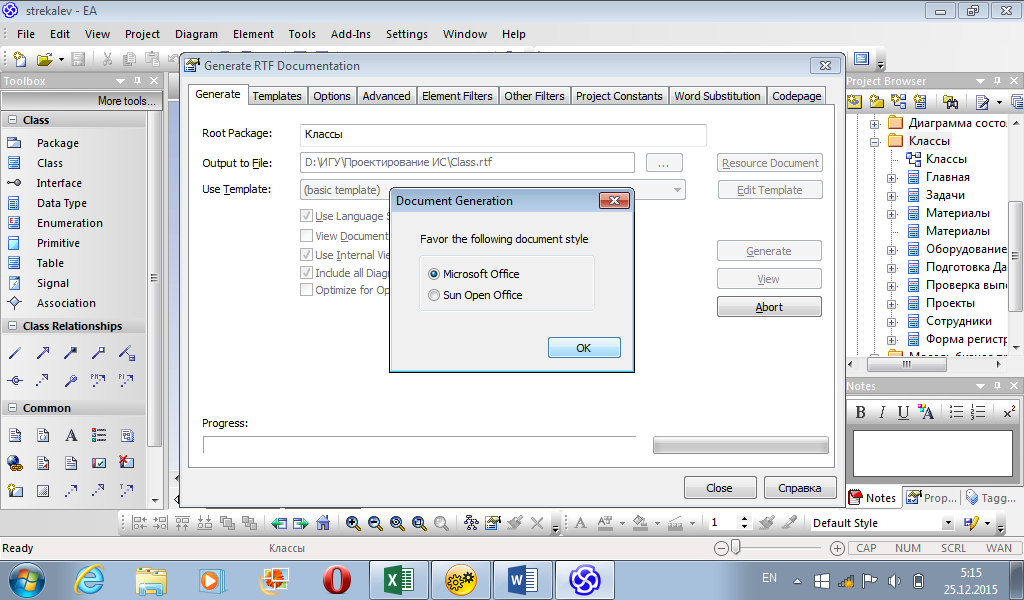
На вкладке Codepage устанавливаем следующие значения (рис. 58):

* Language – 1049 Russian,
* Codepae – 855 Cyrillic,
* Charset – 204 Russian.



**Рис. 58.** Параметры кодировки

На вкладке Generate нажимаем на кнопку Generate. В появившемся окне выбираем Microsoft Office и нажимаем Ok (рис. 59).



**Рис. 59.** Выбор целевой программы создания отчета

В строке Progress отображается процесс построения отчета с индикатором процента выполнения работы. После завершения процесса создания отчета появляется информирующее окно, в котором нажимаем Ok.

В указанном месте сохранения отчета будут созданы два файла: один с отчетом (имя.rtf), а второй содержит выгруженную из Проекта информацию (файл с расширением ldb), который всегда нужно копировать с отчетом, иначе картинки в отчете не будут отображаться.

# Практическое задание по курсу

В рамках курса предлагается выполнить проектирование информационной системы на заданную тему в объеме двух бизнес процессов, последовательно создавая диаграммы в нотации UML, используя инструментальное средство Enterprise Architect и данное методическое руководство. По результатам проектирования сформировать отчет по проекту (о содержании отчета см. приложение).

Список возможных тем для проектирования информационной системы:

1. Библиотека.
2. Поликлиника. Ведение документации.
3. Поликлиника. Ведение личного кабинета.
4. Жилищно-коммунальная компания. Управление жилищным фондом.
5. Жилищно-коммунальная компания. Ведение личного кабинета.
6. Школа.
7. Музыкальная школа.
8. Спортивный клуб.
9. Программа спортивных тренировок.
10. Автосервис.
11. Автомойка.
12. Склад.
13. Магазин.
14. Служба такси.
15. Химчистка.
16. Подбор одежды.
17. Музыкальная картотека.

# Список литературы

1. Буч Г., Максимчук Р., Энгл М., Янг Б., Коналлен Д., Хьюстон К. Объектно-ориентированный анализ и проектирование с примерами приложений. — М. : Вильямс, 2010. — 720 с.
2. Леоненков А.В. Самоучитель UML. — СПБ. : БХВ-Петербург, 2001. — 304 с.
3. Рамбо Дж., Якобсон А., Буч Г. UML: специальный справочник. — СПб. : Питер, 2002. — 656 с.
4. Розенберг Д., Скотт К. Применение объектного моделирования с использованием UML и анализ прецедентов: Пер. с англ. — М. : ДМК Пресс, 2002. — 160 с.
5. Documents Associated With UML® Version 2.3. — URL : http://www.omg.org/spec/UML/2.3/
6. Sparxsystems.com — Сайт разработчика системы Enterprise Architect. — URL : <http://www.sparxsystems.com/>

# Приложение

**Шаблон отчета по проекту**

1. Титульный лист
2. Описание предметной области
3. Постановка задачи
4. Бизнес процессы, предназначенные для автоматизации.
5. Бизнес процессы, предназначенные для автоматизации в рамках курса (2 бизнес процесса).
6. Анализ требований
   1. Требования к системе
   2. Интерфейс системы
   3. Диаграммы деятельности
7. Функции системы
   1. Диаграммы вариантов использования
8. Логическая модель
   1. Диаграммы анализа пригодности
   2. Диаграммы последовательности
   3. Диаграммы классов
   4. Диаграммы деятельности
   5. Диаграммы состояний
9. Физическая модель
   1. Диаграммы компонентов
   2. Диаграммы развертывания
10. Листинг программного кода

О.А. Николайчук

**Инструментальное средство объектно-ориентированного проектирования Enterprise Architect**

Темплан 2017. Поз. 45

Подписано в печать 28.01.2017. Формат 60х84 1/16

Печать трафаретная. Уч.-изд. л. 3 Усл. печ. л. 4

Тираж 100. Заказ 115

ИЗДАТЕЛЬСТВО

ООО «ЦентрНаучСервис»

664033, Иркутск, ул. Лермонтова, 126

1. Данный раздел составлен преподавателем Малтугуевой Г.С. [↑](#footnote-ref-2)