Содержание:

ВВЕДЕНИЕ

Информация в современном мире превратилась в один из наиболее важных ресурсов, а информационные системы (ИС) стали необходимым инструментом практически во всех сферах деятельности. Разнообразие задач, решаемых с помощью ИС, привело к появлению множества разнотипных систем, отличающихся принципами построения и заложенными в них правилами обработки информации.

В настоящее время ни одна организация не обходится без электронно-вычислительных машин и информационных систем, автоматизирующих какой-либо важный процесс.

Проектирование – это процесс составления описания объекта. Это важный этап жизненного цикла разработки программного обеспечения. Описание объекта может быть оформлено в виде текста, таблиц и диаграмм.

Проектирование информационных систем охватывает три основные области:

1. проектирование объектов данных, которые будут реализованы в базе данных;
2. проектирование программ, экранных форм, отчетов;
3. учет конкретной среды и технологии, а именно: топологии сети, конфигурации аппаратных средств и т.д.

Целью исследования является анализ материалов об объектно-ориентированном подходе и обзор наиболее популярных программных средств реализации данного подхода.

Глава 1. Основы объектно-ориентированного подхода к проектированию информационных систем

Сущность объектно-ориентированного подхода

Основной идеей объектно-ориентированного проектирования информационных систем является рассмотрение предметной области и логического решения задачи с точки зрения объектов (понятий или сущностей).

В процессе объектно-ориентированного анализа основное внимание уделяется определению и описанию объектов (или понятий) в терминах предметной области. Например, в случае информационной системы ВУЗа среди понятий должны присутствовать Преподаватель (Lecturer), Студент (Student) и Заведующий кафедрой (Head Of Chair).

В процессе объектно-ориентированного проектирования определяются логические программные объекты, которые будут реализованы средствами объектно-ориентированного языка программирования. Эти программные объекты включают в себя атрибуты и методы.

В процессе конструирования или объектно-ориентированного программирования обеспечивается реализация разработанных компонентов, таких как класс Lecturer на языке C++, C#, Java, Smalltalk или Visual Basic.

Успешность проектов информационных систем в различных отраслях и эффективность их внедрения обусловлены, прежде всего, достаточно строгим определением объектов управления и оптимизируемых бизнес-процессов. Не случайно создание информационной системы производится в контексте общей оптимизации бизнеса и зачастую предполагает реинжиниринг бизнес-процессов предприятия [1].

Базовым средством фиксации (документирования) результатов проектирования систем посредством этих методологий является унифицированный язык моделирования (Unified Modeling Language, UML).

Основные понятия, используемые в объектно-ориентированном подходе

Объектно-ориентированный подход использует объектную декомпозицию, то есть поведение системы описывается в терминах взаимодействия объектов.

Класс - это абстракция множества сущностей реального мира, объединенных общностью структуры и поведения.

Объект - это элемент класса, то есть абстракция определенной сущности.

Необходимо отметить, что объекты активны, у них есть не только внутренняя структура, но и поведение, которое описывается так называемыми методами объекта. Например, может быть определен класс "пользователь", характеризующий "пользователя вообще", то есть ассоциированные с пользователями данные и их поведение (методы).

После этого может быть создан объект "пользователь Иванов" с соответствующей конкретизацией данных и, возможно, методов.

Следующую группу важнейших понятий объектного подхода составляют инкапсуляция, наследование и полиморфизм.

Основным инструментом борьбы со сложностью в объектно-ориентированном подходе является инкапсуляция - сокрытие реализации объектов (их внутренней структуры и деталей реализации методов) с предоставлением во вне только строго определенных интерфейсов.

Понятие " полиморфизм " может трактоваться как способность объекта принадлежать более чем одному классу. Введение этого понятия отражает необходимость смотреть на объекты под разными углами зрения, выделять при построении абстракций разные аспекты сущностей моделируемой предметной области, не нарушая при этом целостности объекта. (Строго говоря, существуют и другие виды полиморфизма, такие как перегрузка и параметрический полиморфизм, но нас они сейчас не интересуют.)

Наследование означает построение новых классов, на основе существующих, с возможностью добавления или переопределения данных и методов. Наследование является важным инструментом борьбы с размножением сущностей без необходимости. Общая информация не дублируется, указывается только то, что меняется. При этом класс -потомок помнит о своих "корнях". [2]

Базовые составляющие объектно-ориентированного подхода

Базовыми составляющими объектно-ориентированного подхода являются:

а) унифицированный процесс;

б) унифицированный язык моделирования;

в) шаблоны проектирования.

Унифицированный процесс – это процесс разработки программного обеспечения (ПО), который обеспечивает упорядоченный подход к распределению задач и обязанностей в организации-разработчике. Унифицированный процесс охватывает весь жизненный цикл ПО, начиная с определения требований и заканчивая сопровождением, и представляет собой обобщенный каркас (шаблон, скелет), который может быть применен (специализирован) для разработки и сопровождения широкого круга систем.

Неотъемлемой частью унифицированного процесса является UML – язык (система обозначений) для определения, визуализации и конструирования моделей системы в виде диаграмм и документов на основе объектно-ориентированного подхода. Следует отметить, что Унифицированный процесс и UML разрабатывались совместно.

На стадиях анализа и проектирования часто используются так называемые шаблоны (паттерны) проектирования. Шаблон – это именованная пара «проблема/решение», содержащая готовое обобщенное решение типичной проблемы. Как правило, шаблон помимо текстового описания содержит также одну или несколько диаграмм UML (например, диаграммы классов, последовательности и/или коммуникации), графически иллюстрирующих состав и структуру классов, а также особенности их взаимодействия при решении поставленной проблемы. Шаблоны разрабатываются опытными профессионалами и являются проверенными, эффективными (порой оптимальными) решениями. Применение шаблонов может резко сократить затраты и повысить качество разработки ПО. [3]

Преимущества объектно-ориентированного подхода

В отличие от структурного подхода объектно-ориентированный имеет ряд преимуществ:

1) описание системы в виде объектов больше соответствует содержательному смыслу предметной области. Например, при использовании структурного подхода БД должна удовлетворять требованиям нормализации, в соответствии с которыми данные по одному и тому же объекту (сущности из реального мира) могут храниться в нескольких таблицах;

2) сущности реального мира, как правило, обладают поведением, что в объектно-ориентированном проектировании отражается с помощью определения методов класса. В структурном подходе данные (атрибуты) и алгоритмы (методы)

существуют отдельно друг от друга;

3) объединение атрибутов и методов в объекте (классе), а также инкапсуляция позволяют добиться большей внутренней и меньшей внешней связности между компонентами системы. Это облегчает решение проблем:

а) адаптации системы к изменению существующих или появлению новых требований;

б) сопровождения системы на разных стадиях жизненного цикла;

в) повторного использования компонентов.

4) объектно-ориентированный подход позволяет легче организовать параллельные вычисления, так как каждый объект обладает собственными значениями характеристик (атрибутов) и поведением, за счет чего можно добиться его автономной работы;

5) Case-средства, поддерживающие объектно-ориентированный подход, на основе информации об объектах позволяют достичь большей степени автоматизации кодогенерации. Case-средства, поддерживающие структурный подход, хорошо справляются с генерацией структур БД. Однако следует отметить, что эта структура должна удовлетворять требованиям нормализации. В связи с чем автоматическая кодогенерация (например, экранов или функций обработки данных) возможна лишь в редких случаях. может резко сократить затраты и повысить качество разработки ПО. [4]

Язык проектирования UML

Все диаграммы UML можно условно разбить на две группы, первая из которых – общие диаграммы. Общие диаграммы практически не зависят от предмета моделирования и могут применяться в любом программном проекте без оглядки на предметную область, область решений и т.д.

Диаграмма вариантов использования (use case diagram) – это наиболее общее представление функционального назначения системы.

Диаграмма вариантов использования призвана ответить на главный вопрос моделирования: что делает система во внешнем мире?

На диаграмме вариантов использования применяются два типа основных сущностей: варианты использования 1 и действующие лица 2, между которыми устанавливаются следующие основные типы отношений:

1. ассоциация между действующим лицом и вариантом использования 3;
2. обобщение между действующими лицами 4;
3. обобщение между вариантами использования 5;
4. зависимости (различных типов) между вариантами использования 6.

На диаграмме использования, как и на любой другой, могут присутствовать комментарии 7. Более того, это настоятельно рекомендуется делать для улучшения читаемости диаграмм. Основные элементы нотации представлены на рисунке 1.

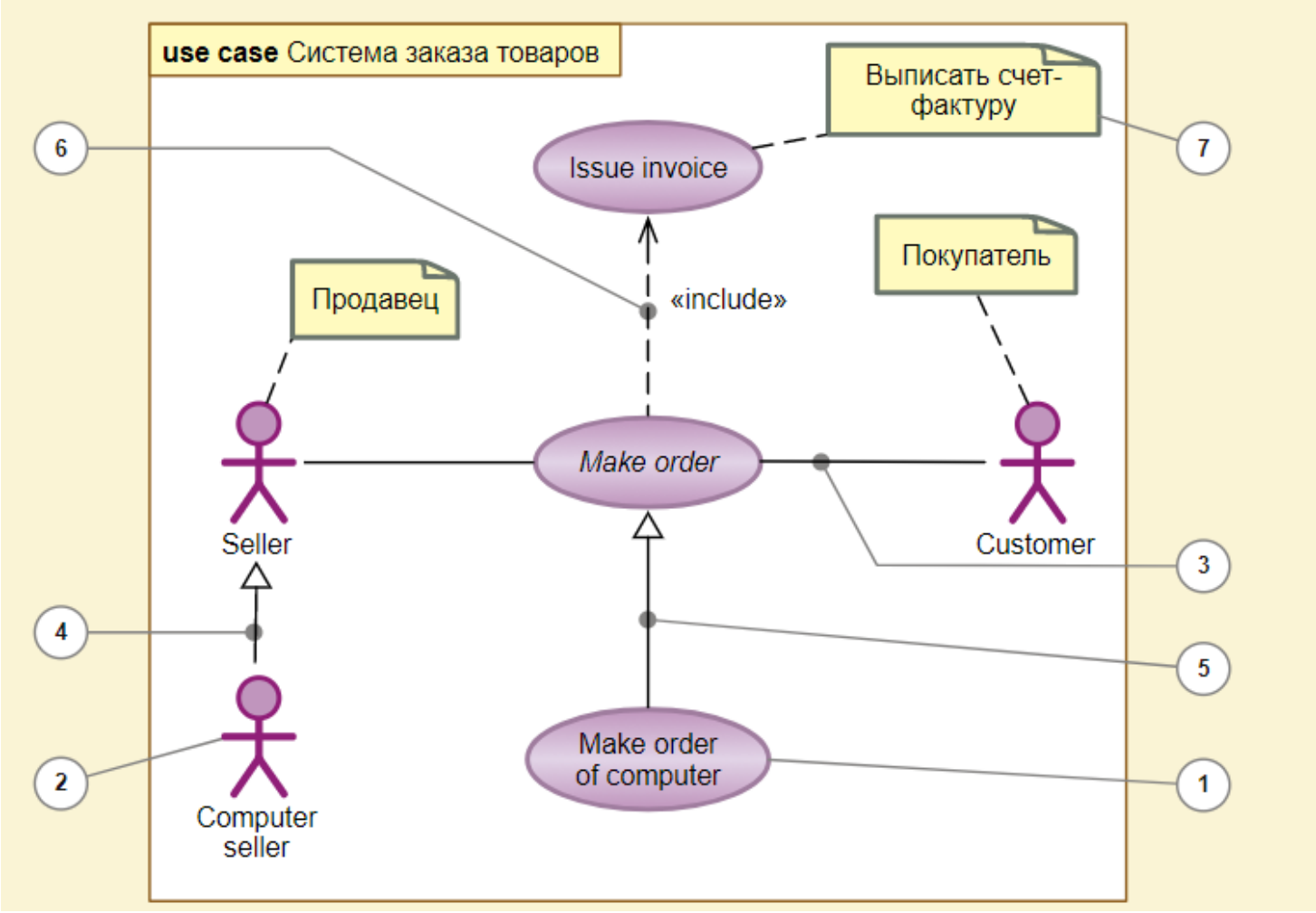

Рисунок 1. Нотация диаграммы вариантов использования

Диаграмма классов (class diagram) – основной способ описания структуры системы.

На диаграмме классов применяется один основной тип сущностей:
классы 1 (включая многочисленные частные случаи классов: интерфейсы, примитивные типы, классы-ассоциации и многие другие), между которыми устанавливаются следующие основные типы отношений:

1. ассоциация между классами (с множеством дополнительных подробностей);
2. обобщение между классами;
3. зависимости (различных типов).

Некоторые элементы нотации, применяемые на диаграмме классов, показаны на рисунке 2.

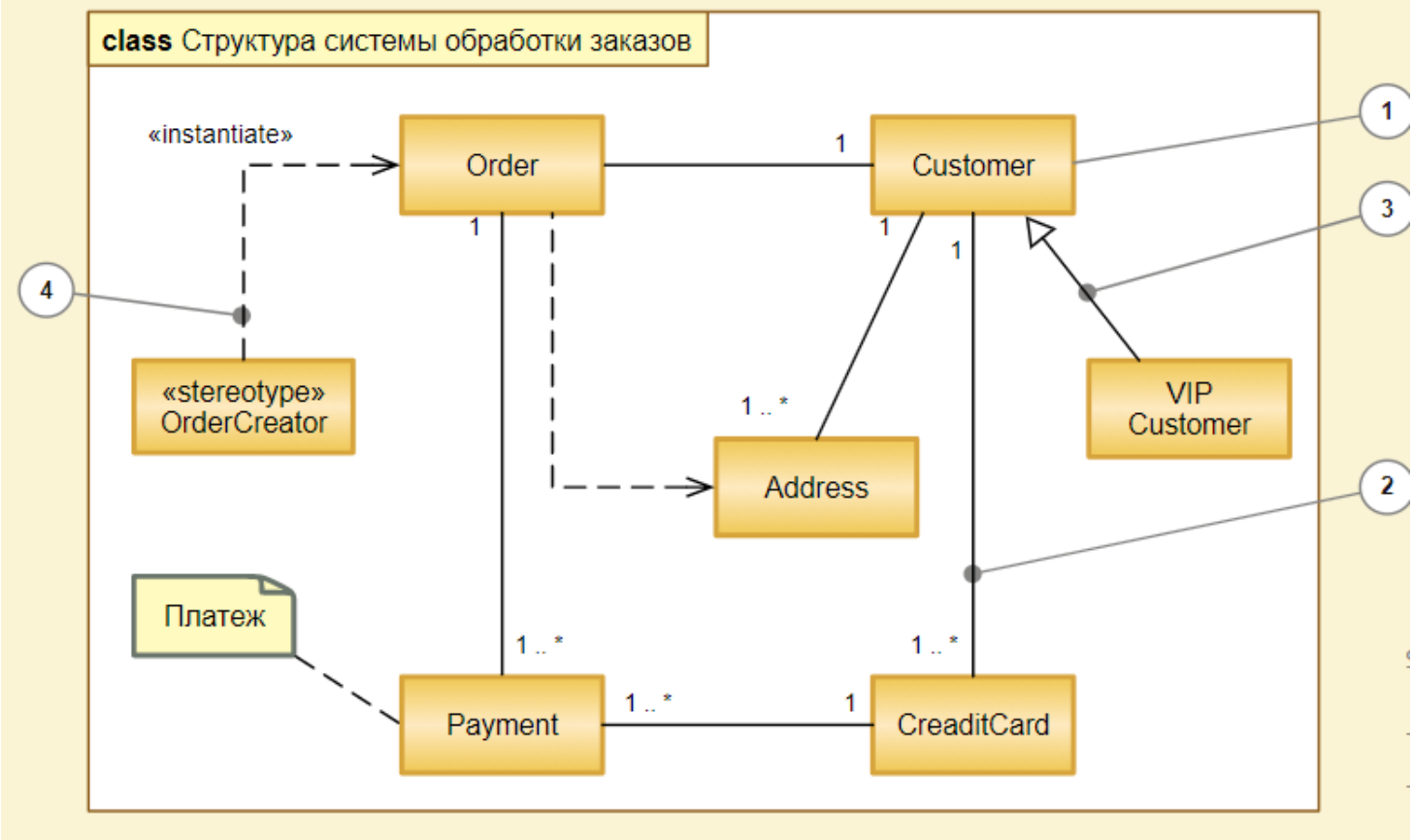

Рисунок 2. Нотация диаграммы классов

Диаграмма автомата (state machine diagram) – это один из способов детального описания поведения в UML на основе явного выделения состояний и описания переходов между состояниями.

В сущности, диаграммы автомата, как это следует из названия, представляют собой граф переходов состояний, нагруженный множеством дополнительных деталей и подробностей.

На диаграмме автомата применяют один основной тип сущностей – состояния 1, и один тип отношений – переходы 2, но и для тех и для других определено множество разновидностей, специальных случаев и дополнительных обозначений. (см. рисунок 3)

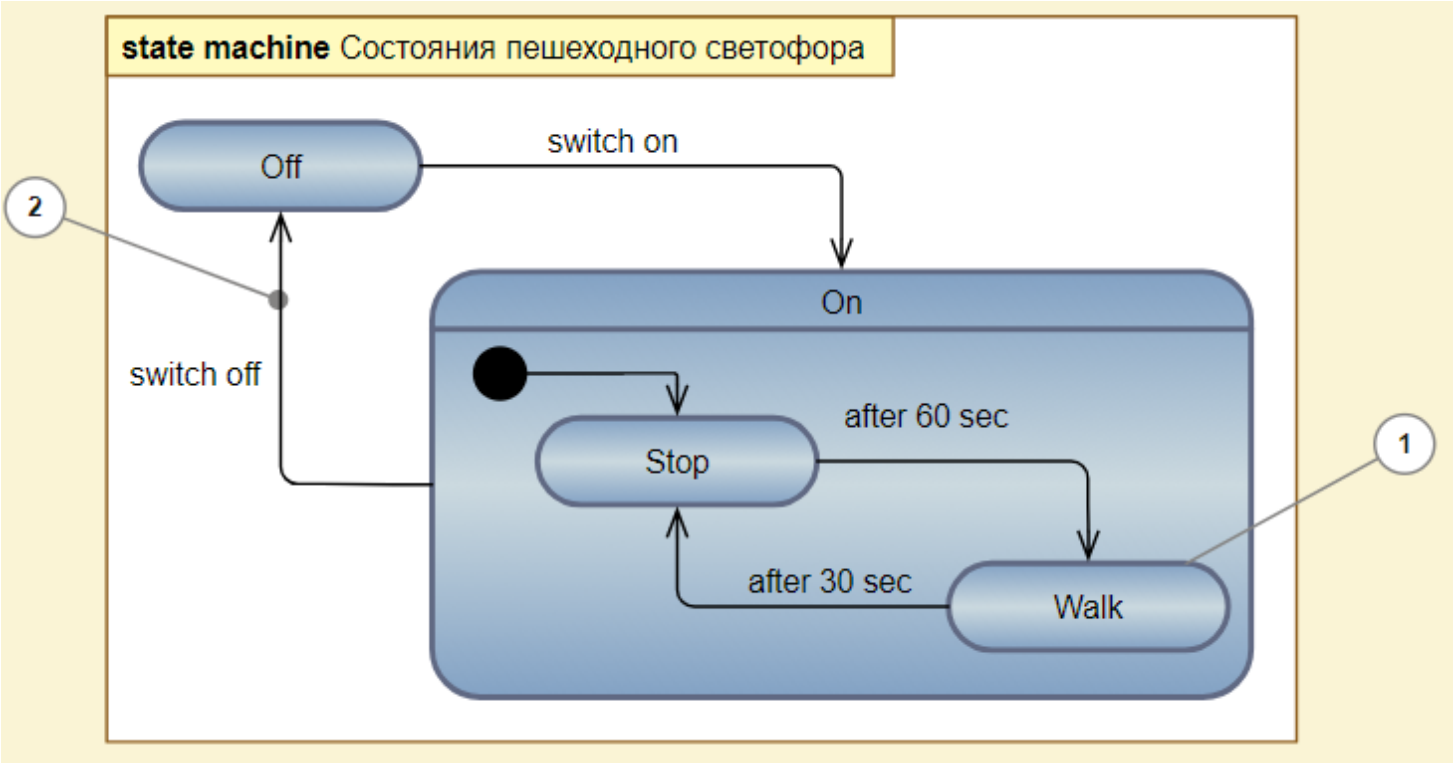

Рисунок 3. Нотация диаграммы автомата

Диаграмма деятельности (activity diagram) – способ описания поведения на основе указания потоков управления и потоков данных.

Диаграмма деятельности – еще один способ описания поведения, который визуально напоминает старую добрую блок-схему алгоритма. Однако за счет модернизированных обозначений, согласованных с объектно-ориентированным подходом, а главное, за счет новой семантической составляющей (свободная интерпретация сетей Петри), диаграмма деятельности UML является мощным средством для описания поведения системы.

На диаграмме деятельности применяют один основной тип сущностей – действие 1, и один тип отношений – переходы 2 (передачи управления и данных). Также используются такие конструкции как развилки, слияния, соединения, ветвления 3, которые похожи на сущности, но таковыми на самом деле не являются, а представляют собой графический способ изображения некоторых частных случаев многоместных отношений. Основные элементы нотации,

применяемые на диаграмме деятельности, показаны на рисунке 4.

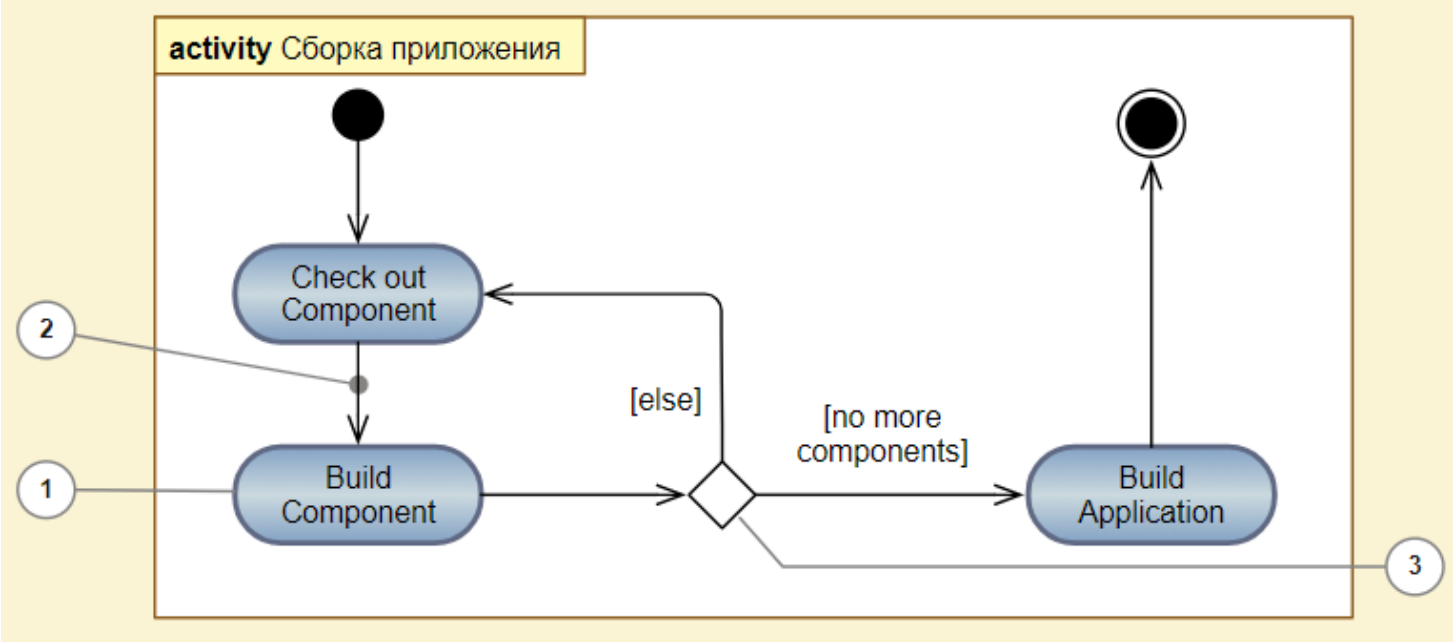

Рисунок 4. Нотация диаграммы деятельности

Диаграмма последовательности (sequence diagram) – это способ описания поведения системы на основе указания последовательности передаваемых сообщений.

Фактически, диаграмма последовательности – это запись протокола конкретного сеанса работы системы (или фрагмента такого протокола). В объектно-ориентированном программировании самым существенным во время выполнения является пересылка сообщений между взаимодействующими объектами. Именно последовательность посылок сообщений отображается на данной диаграмме, отсюда и название.

На диаграмме последовательности применяют один основной тип сущностей – экземпляры взаимодействующих классификаторов 1 (в основном классов, компонентов и действующих лиц), и один тип отношений – связи 2, по которым происходит обмен сообщениями 3. Предусмотрено несколько способов посылки сообщений, которые в графической нотации различаются видом стрелки, соответствующей отношению.

Важным аспектом диаграммы последовательности является явное отображение течения времени. В отличие от других типов диаграмм, кроме разве что диаграмм синхронизации, на диаграмме последовательности имеет значение не только наличие графических связей между элементами, но и взаимное расположение

элементов на диаграмме. А именно, считается, что имеется (невидимая) ось времени, по умолчанию направленная сверху вниз, и то сообщение, которое отправлено позже, нарисовано ниже.

На рисунке 5 показаны основные элементы нотации, применяемые на диаграмме последовательности. Для обозначения самих взаимодействующих объектов применяется стандартная нотация – прямоугольник с именем экземпляра классификатора. Пунктирная линия, выходящая из него, называется линией жизни (lifeline) 4. Это не обозначение отношения в модели, а графический комментарий, призванный направить взгляд читателя диаграммы в правильном направлении. Фигуры в виде узких полосок, наложенных на линию жизни, также не являются изображениями моделируемых сущностей. Это графический комментарий, показывающий отрезки времени, в течении которых объект владеет потоком управления (execution occurrence) 5 или другими словами имеет место активация(activation) объекта. Составные шаги взаимодействия (combined fragment) 6 позволяют на диаграмме последовательности, отражать и алгоритмические аспекты протокола взаимодействия.

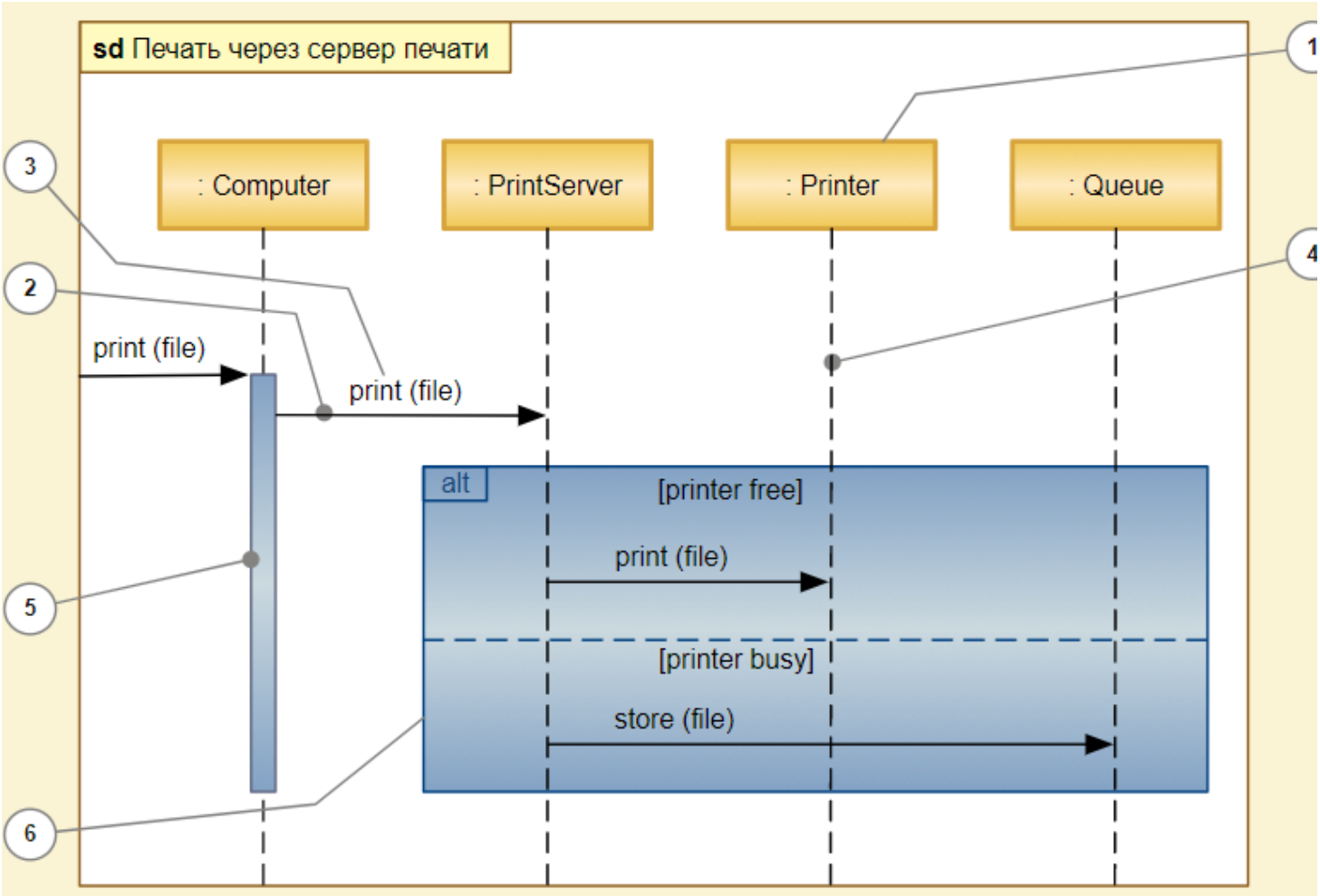

Рисунок 5. Нотация диаграммы последовательности

Диаграмма коммуникации (communication diagram) – способ описания поведения, семантически эквивалентный диаграмме последовательности.

Фактически, это такое же описание последовательности обмена сообщениями взаимодействующих экземпляров классификаторов, только выраженное другими графическими средствами. Более того, большинство инструментов умеет автоматически преобразовывать диаграммы последовательности в диаграммы коммуникации и обратно.

Таким образом, на диаграмме коммуникации также, как и на диаграмме последовательности, применяют один основной тип сущностей – экземпляры взаимодействующих классификаторов 1 и один тип отношений – связи 2. Однако здесь акцент делается не на времени, а на структуре связей между конкретными экземплярами.

На рисунке 6 показаны основные элементы нотации, применяемые на диаграмме коммуникации. Для обозначения самих взаимодействующих объектов применяется стандартная нотация – прямоугольник с именем экземпляра классификатора. Взаимное положение элементов на диаграмме кооперации не имеет значения – важны только связи (чаще всего экземпляры ассоциаций), вдоль которых передаются сообщения 3. Для отображения упорядоченности сообщений во времени применяется иерархическая десятичная нумерация.

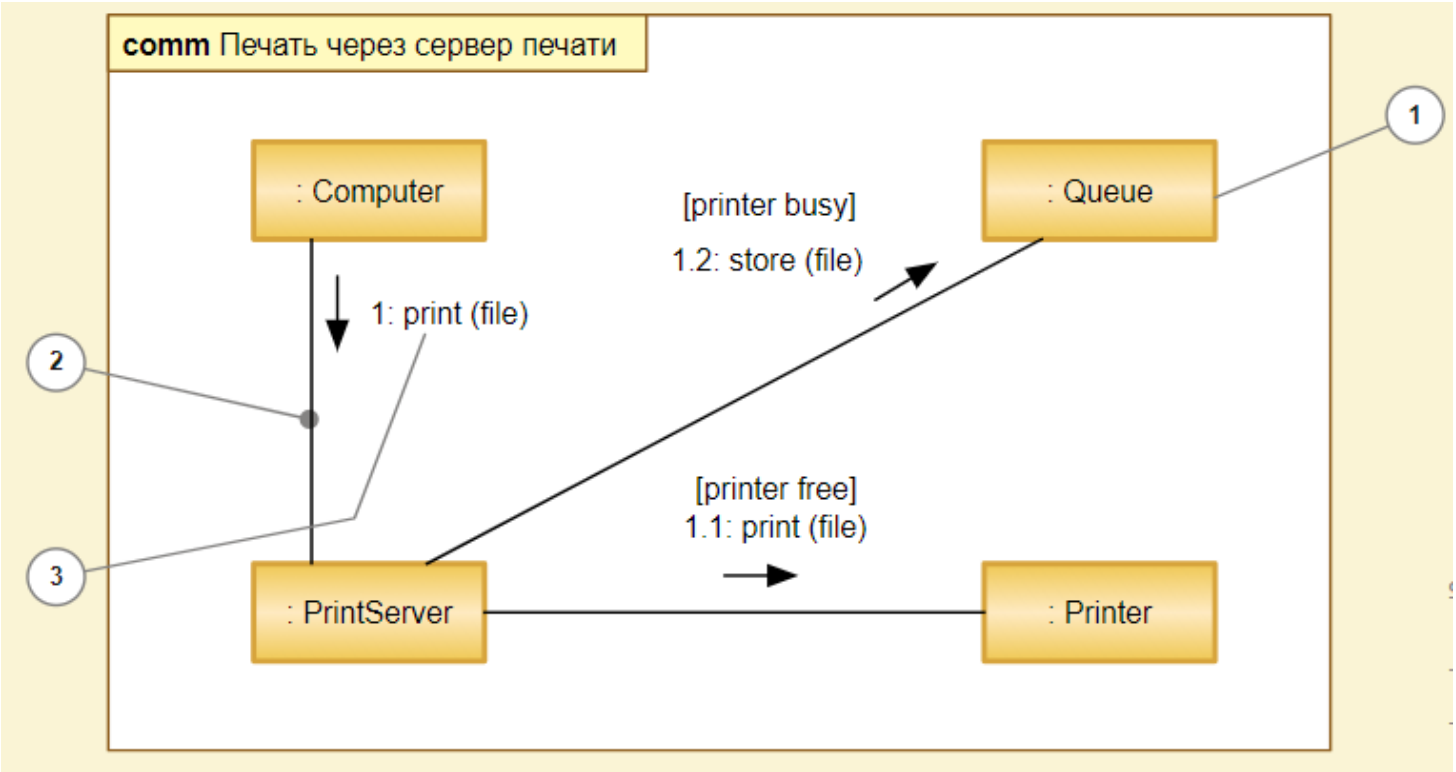

Рисунок 6. Нотация диаграммы коммуникации

Диаграмма компонентов (component diagram) – показывает взаимосвязи между модулями (логическими или физическими), из которых состоит моделируемая система.

Основной тип сущностей на диаграмме компонентов – это сами компоненты 1, а также интерфейсы 2, посредством которых указывается взаимосвязь между компонентами. На диаграмме компонентов применяются следующие отношения:

1. реализации между компонентами и интерфейсами (компонент реализует интерфейс);
2. зависимости между компонентами и интерфейсами (компонент использует интерфейс) 3.

На рисунке 7 показаны основные элементы нотации, применяемые на диаграмме компонентов.

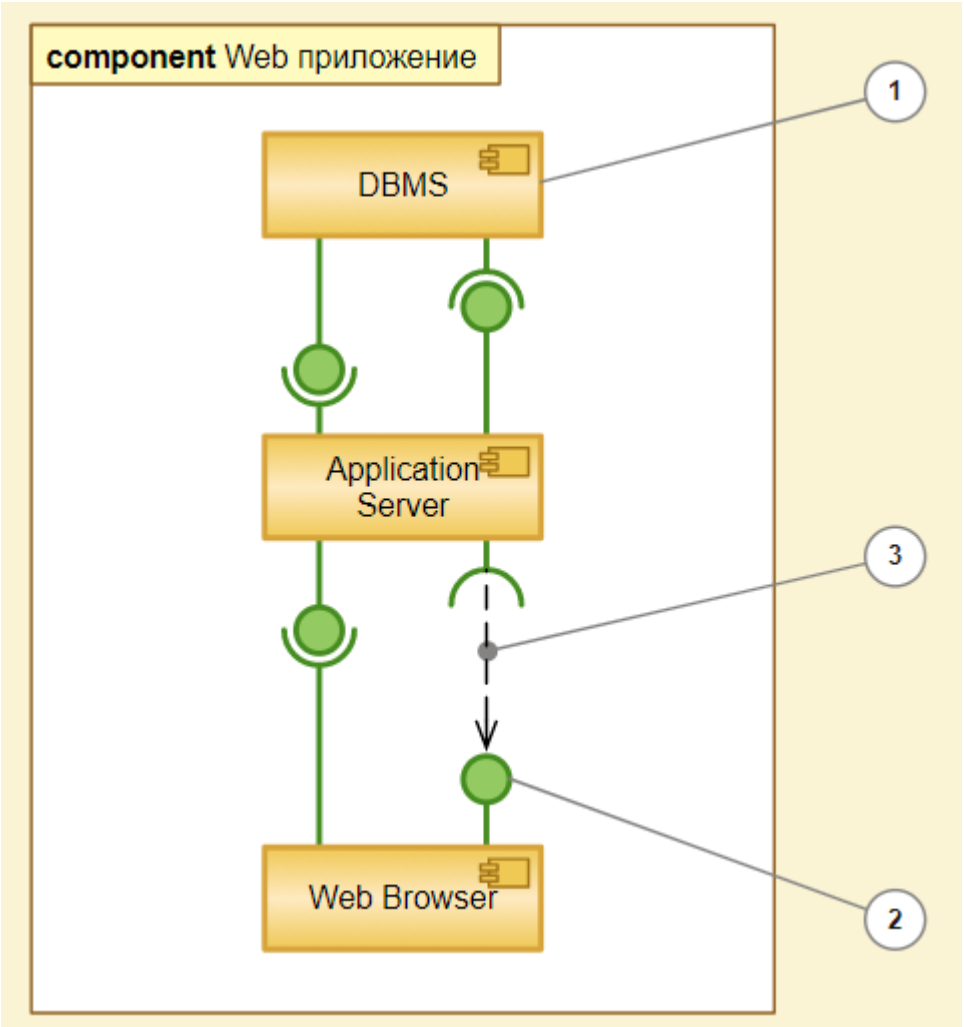

Рисунок 7. Нотация диаграммы компонентов

Диаграмма размещения (deployment diagram) наряду с отображением состава и связей элементов системы показывает, как они физически размещены на вычислительных ресурсах во время выполнения.

Таким образом, на диаграмме размещения, по сравнению с диаграммой компонентов, добавляется два типа сущностей: артефакт 1, который является реализацией компонента 2 и узел 3 (может быть, как классификатор, описывающий тип узла, так и конкретный экземпляр), а также отношение ассоциации между узлами 4, показывающее, что узлы физически связаны во время выполнения.

На рисунке 8 показаны основные элементы нотации, применяемые на диаграмме размещения. Для того чтобы показать, что одна сущность является частью другой, применяется либо отношение зависимости «deploy» 5, либо фигура одной сущности помещается внутрь фигуры другой сущности 6. [4]

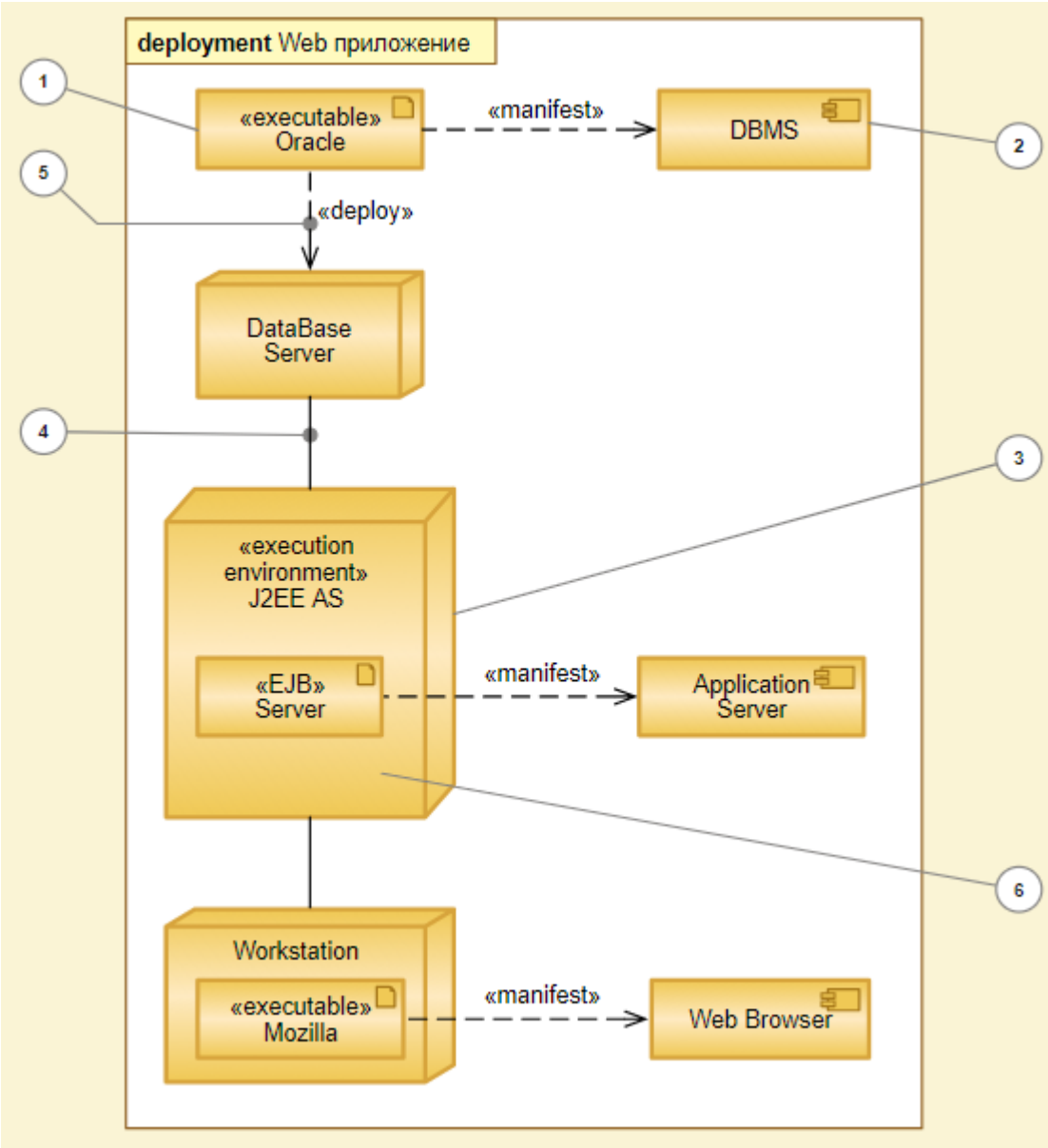

Рисунок 8. Нотация диаграммы размещения

UML объектно-ориентирован, в результате чего методы описания результатов анализа и проектирования семантически близки к методам программирования на современных объектно-ориентированных языках. Язык UML позволяет описать систему практически со всех возможных точек зрения и разные аспекты поведения системы.

Диаграммы UML сравнительно просты для чтения после достаточно быстрого ознакомления с его синтаксисом. UML получил широкое распространение и динамично развивается.

Глава 2. Средства реализации объектно-ориентированного подхода при проектировании информационных систем

Sparx Systems Enterprise Architect

Enterprise Architect - это программа для UML-моделирования и проектирования, с возможностью многопользовательской работы и дружественным интерфейсом. Enterprise Architect работает на Windows и Linux.

Ниже перечислены одни из возможностей программы для проектирования Enterprise Architect весьма многочисленны:

1. нотация UML 2.0 с поддержкой всех видов диаграмм;
2. поддержка C++, Java, C#, VB, VB.Net, Delphi, PHP, .NET;
3. моделирование БД, прямое проектирование в DDL и обратное проектирование из ODBC;
4. загружаемые UML-профили (например, SPEM), позволяющие создавать узкоспециализированные модели;
5. поддержка паттернов проектирования;
6. генерация документации в форматах HTML и RTF;
7. многопользовательская работа, утилиты для менеджера проекта, тестирование, глоссарий, другие ресурсы;
8. автоматизация интерфейса, поддержка макросов.

Enterprise Architect существует в трех редакциях:

a) EA Desktop Edition

Интуитивно понятная утилита для UML-моделирования, предназначенная для индивидуальных аналитиков и/или разработчиков. Простейший инструмент проектирования, имеющий некоторые ограничения. Отсутствуют многие, привычные для профессионалов, функции, которые, впрочем, абсолютно не нужны, если вы просто ищете инструмент для рисования UML-диаграмм. Не поддерживается, например, импорт/экспорт кода и DDL, Active X-интерфейс и совместный доступ к диаграммам.

б) EA Professional Edition

Полнофункциональная среда UML-моделирования, нацеленная на групповую разработку, поддерживает совместный доступ к созданным моделям, Active X, *XMI*, импорт/экспорт кода и DDL, извлечение схем БД Oracle, SQL Server и MS Access.

в) EA Corporate Edition

Наиболее полная редакция, включающая все возможности настольной и профессиональной версий плюс возможность соединения с MySQL, SQL Server, PostgreSQL, Sybase Adaptive Server Anywhere и Oracle9i. Также эта редакция поддерживает авторизацию пользователей, группы пользователей, блокировку элементов. Эта версия предназначена для больших команд.

На рисунке 9 представлен интерфейс программы Enterprise Architect.

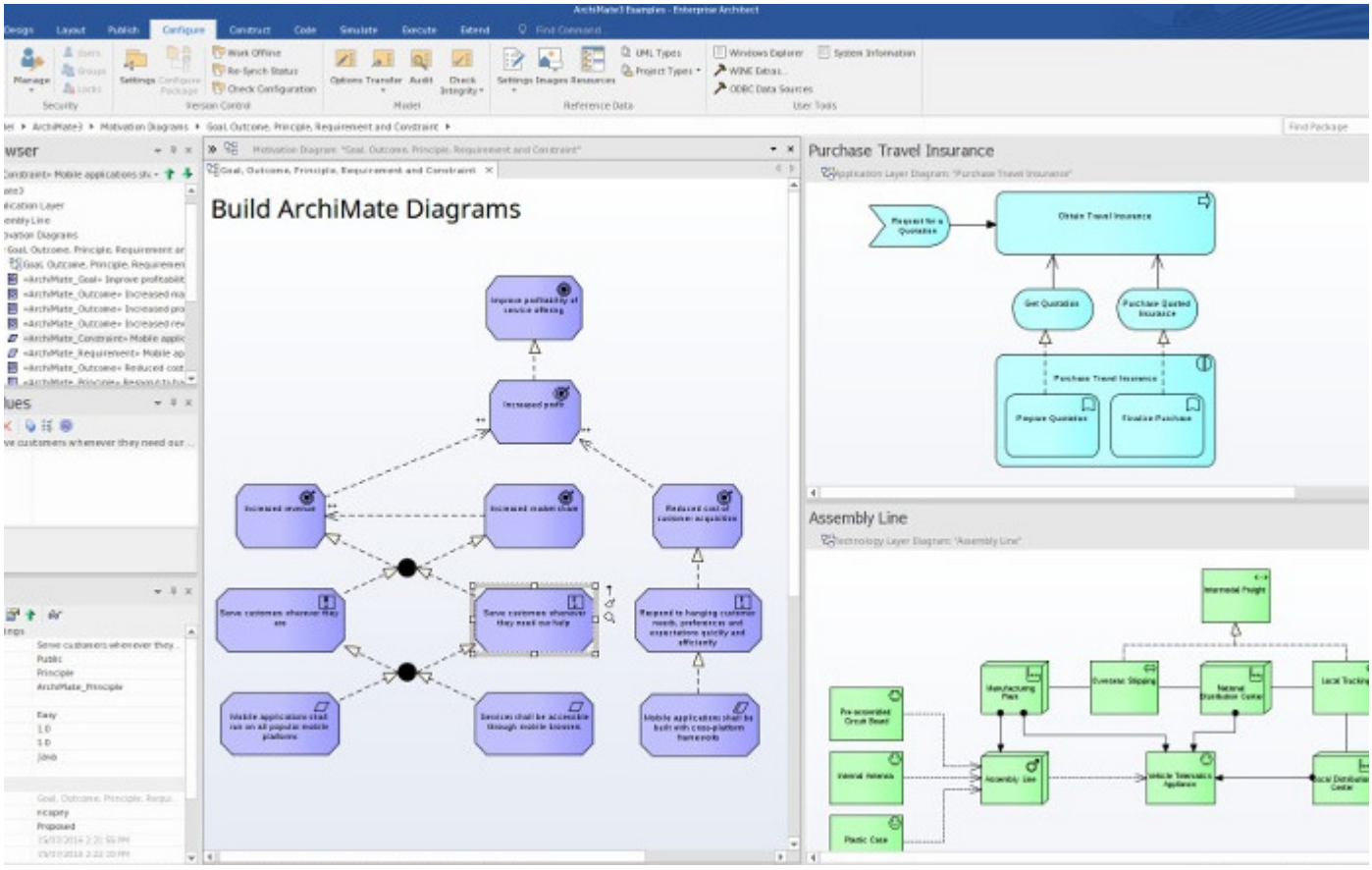

Рисунок 9. Интерфейс программы Enterprise Architect

Существует версия EA Lite FREE read-only. Она позиционируется как решение, позволяющее зарегистрированным пользователям демонстрировать UML-диаграммы персоналу и заказчикам. EA Lite имеет все базовые функции Enterprise

Architect, за исключением создания документации и сохранения. По сути, это вьюер файлов Enterprise Architect.

С EA отлично интегрируется другой продукт Sparx Systems - Zicom Mentor. Zicom Mentor - это ПО для обучения UML, которое поможет пользователям мгновенно получить ответы на свои вопросы, получить и проверить знание UML, начать новый UML-проект.

Zicom Mentor включает интерактивные курсы и тесты, документацию и справочные материалы по UML, а также визуальный словарь UML, справочник по диаграммам и символам и др. [5]

Visual paradigm

Visual Paradigm Suite - это набор средств для моделирования информационных систем и бизнес-процессов, генерации кода на базе построенных моделей, проектирования БД и решения многих других задач.

В пакет Visual Paradigm Suite 5.0 входят следующие инструменты:

1. Visual Paradigm for UML 8.0
2. Business Process Visual Architect 4.0
3. Visual Paradigm SDE
4. Agilian 3.0
5. Database Visual Architect 6.0

VP for UML – это система управления требованиями, поддерживающая полный цикл разработки программного продукта – анализ, дизайн архитектуры, разработка программного кода, тестирование и размещение продукта на стороне заказчика. VP также обеспечивает поддержку версионности и одновременной работы команды пользователей над одним проектом.

В целом, VP for UML является прямым конкурентом небезызвестного Enterprise Architect от компании Sparx Systems.

VP for UML предоставляет широкие возможности для создания и организации требований. Для создания моделей аналитик в рамках одного проекта может описывать требования, одновременно используя такие стандарты и нотации моделирования, как UML, BMPN, OMG, Mind Maps и ArchiMate. Это является большим плюсом, так как, используя VP for UML, не нужно переключаться между большим количеством программ для создания моделей в различных нотациях, а описание

требований различными методами и нотациями должно лишь облегчить понимание проекта.

Функциональные возможности системы:

1. **Построение диаграмм**

Моделирование диаграмм реализовано на довольно высоком уровне, и все элементы интуитивно находятся под рукой. Имеется панель инструментов, на которой располагаются элементы для выбранного типа диаграммы (Use Case, Activity, Mind Map и т.д.). Добавление новых элементов можно осуществлять из «всплывающего» меню, которое становится доступным при выборе любого элемента, уже добавленного на диаграмму, что значительно сокращает время, затрачиваемое на создание диаграммы. (см. рисунок 10)

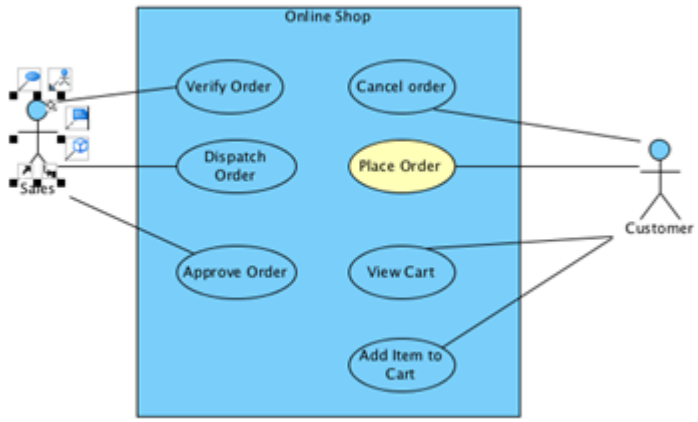

Рисунок 10. Диаграмма вариантов использования в программе Visual Paradigm

Также необходимо отметить две дополнительные функции, которых не хватает в большинстве других системах для Use Case моделирования – Magnet и Sweeper. На рисунке 11 представлена часть инструментов ситсемы.

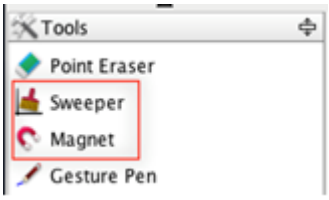

Рисунок 11. Инструменты Magnet и Sweeper

Sweeper раздвигает элементы, создавая тем самым больше рабочего пространства для добавления новых элементов на диаграмму. **Magnet** позволяет сократить пространство между элементами диаграммы, что является весьма удобным при экспорте диаграммы в текстовый документ.

Добавление спецификации к элементам диаграммы существенно напоминает Enterprise Architect, правда у актеров, выполняющих действие, отсутствует разделение по типам (Actor, System). (см. рисунок 12)

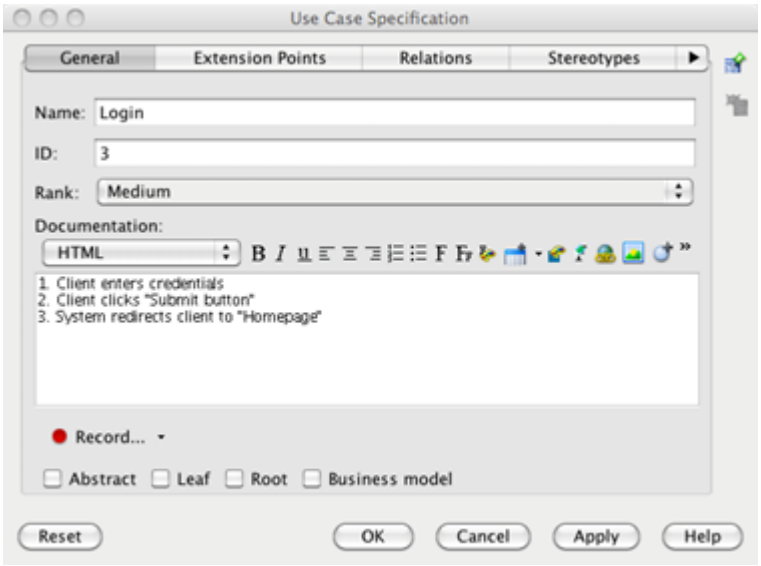

Рисунок 12. Спецификация диаграммы Use Case

1. **Текстовый анализ**

Один из видов поддерживаемых диаграмм — «Текстовый анализ». В режиме текстового анализа предполагается ввод User Stories в определенную область программы, после чего пользователь имеет возможность добавления различных элементов диаграммы путем выделения соответствующего слова/фразы из User Story и определения типа элемента (Use Case, Class, Activity, User и т.д.) (см. рисунок 13).

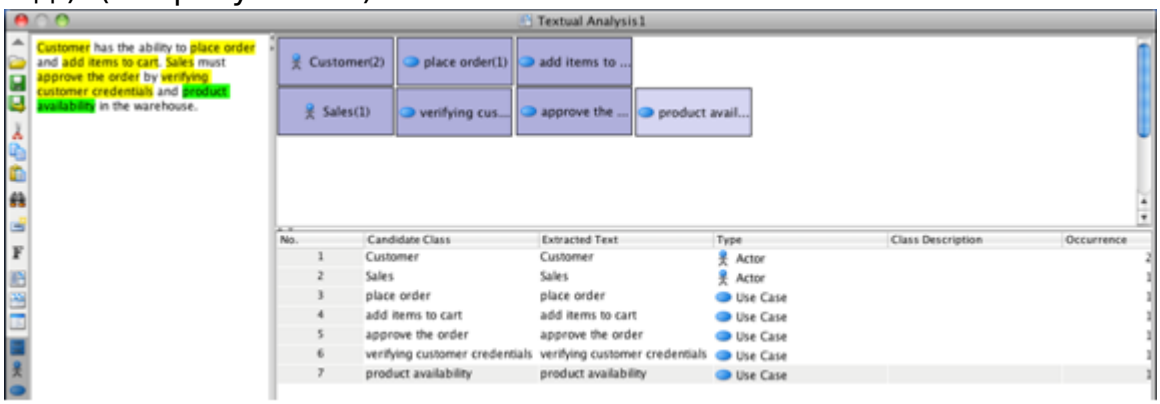

Рисунок 13. Текстовый анализ

После добавления информации в режиме текстового анализа все элементы доступны для добавления в диаграммы любого другого вида при условии, что добавление этих элементов не противоречит нотации. Чтобы просмотреть элементы, определенные в режиме текстового анализа, необходимо нажать иконку «Model Explorer» в браузере проекта.

1. **Моделирование бизнес процессов с помощью BPMN**
   В VP for UML предусмотрена возможность построения диаграмм с помощью нотации BPMN. В целом, построение BPD (Business Process Diagram) ничем не отличается от построения UML-диаграмм. (см. рисунок 14) Следует отметить приятное оформление самой нотации:

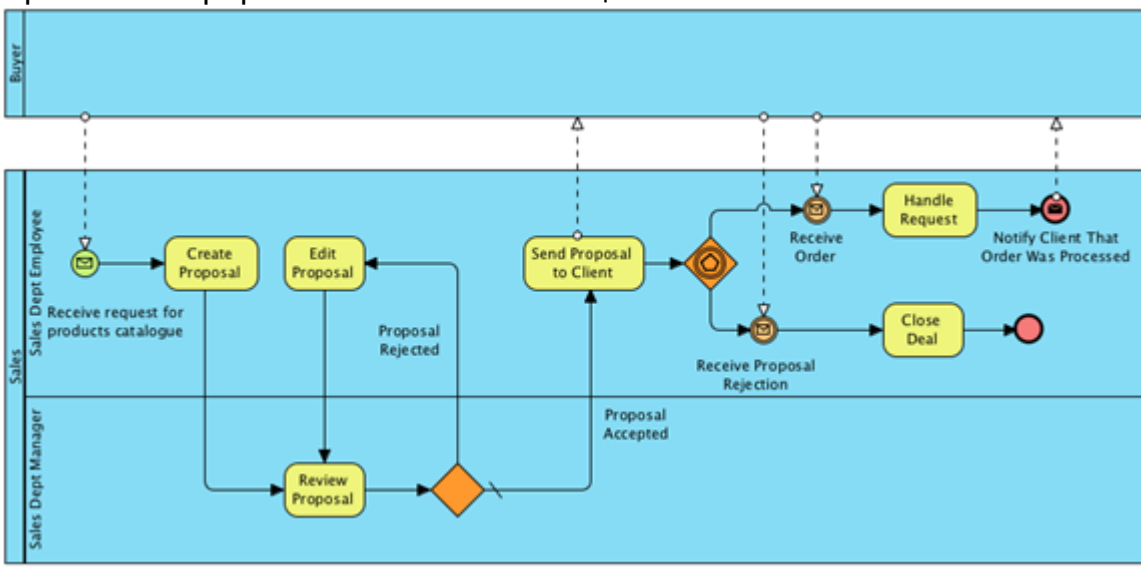

Рисунок 14. Нотация BPMN в программе Visual Paradigm

Также следует выделить возможность генерации кода из BPD в следующие языки определения и исполнения бизнес-процессов:

BPEL – Business Process Execution Language

WSDL – Web Services Description Language

XPDL – XML Process Definition Language

JPDL – jBPM Process Definition Language

1. **Пользовательский интерфейс**

В целом, интерфейс VP for UML не вызывает путаницы и недопонимания. Единственным потенциальным минусом (а возможно и плюсом) является огромное количество настроек для каждого элемента диаграммы и приложения в целом – поначалу в них довольно просто запутаться.

1. **Командная работа**

VP for UML предоставляет широкие возможности для совместной работы пользователей над одним проектом. При этом используются различные «ветки», изменения в которых впоследствии объединяются в главной «ветви» — «trunk». VP for UML позволяет различным членам команды работать над одной диаграммой и сравнивать изменения, внесенные пользователями на разных этапах проекта. При обнаружении конфликта (например, если два пользователя сделают изменения в одной и той же диаграмме, которые будут конфликтовать между собой) программа покажет соответствующее сообщение и не даст внести изменения до тех пор, пока конфликт не будет разрешен.

При совместной работе имеется возможность определения различных ролей на проекте и их прав доступа к проекту. Также можно ограничить доступ к определенным диаграммам путем задания пароля.

Еще одним важным моментом является возможность создание «тэгов» — «снэпшотов» системы. При создании «тэга» фактически создается копия проекта, в которую нельзя вносить изменения. Эта функциональность может быть полезна для проведения «sign-off» с клиентами и для определения объемов работ для каждого конкретного релиза.

Поддерживаются следующие режимы совместной работы:

1. VP Teamwork Server
2. Subversion (SVN)
3. Perforce
4. CVS
5. **Генерация отчетов**

Генерация отчетов возможна в форматах PDF, HTML и MS Word, с возможностью публикации отчетов на сервере.

Генерируемые отчеты получаются довольно громоздкими, так как VP for UML добавляет спецификацию для каждого элемента диаграммы.

1. **Impact Analysis (анализ воздействий)**

Impact Analysis подразумевает то, что можно оценить последствия изменений, вносимых в уже созданные модели. Делается данный анализ при помощи построения «матрицы связей», которая отображает связи между выбранными элементами.

На рисунке 15 ниже представлены матрицы со следующими связями: Use Cases со связью типа «Includes», Use Cases и Actors со связью типа «Association» и Classes со связью «Association».

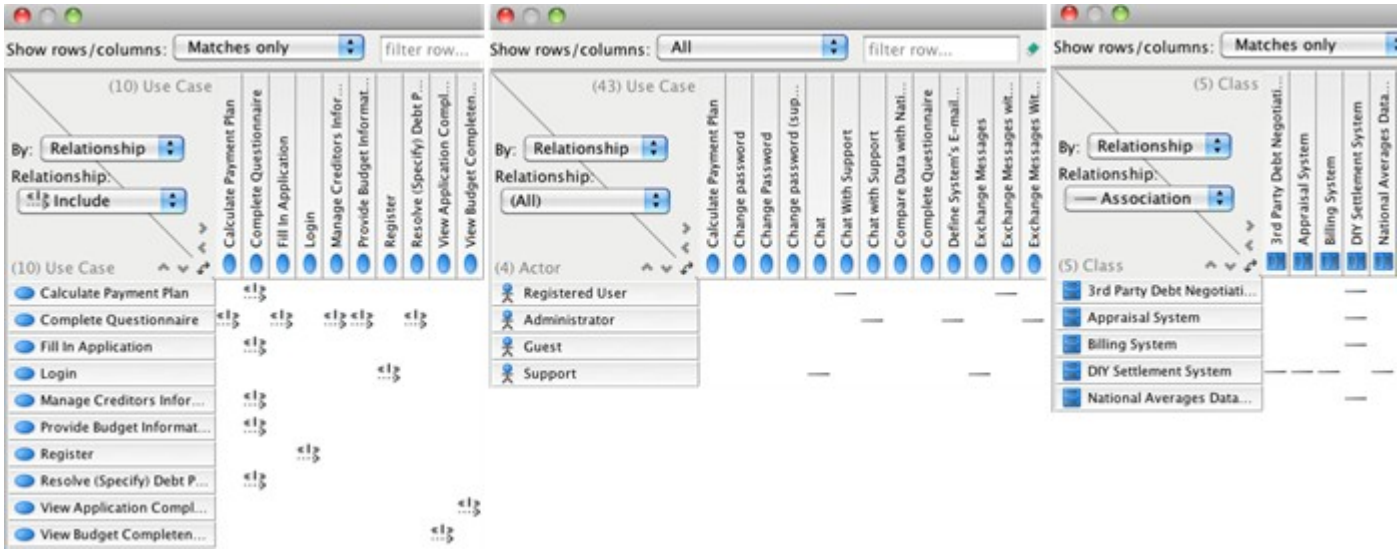

Рисунок 15. Матрицы

1. Animatian and Simulatian

Отдельно хотелось бы выделить функции “Animatian” и “Simulatian”, предназначенные для моделирования и симулирования процессов.

Animatian позволяет «пробежаться» по одному из выбранных потоков диаграммы. Результат можно экспортировать в формат FLV и демонстрировать заказчикам или членам проектной команды.

Также при помощи данного функционала можно выявить неверные потоки и внести соответствующие изменения, что может в итоге сохранить большое количество времени при проектировании. (см. рисунок 16)

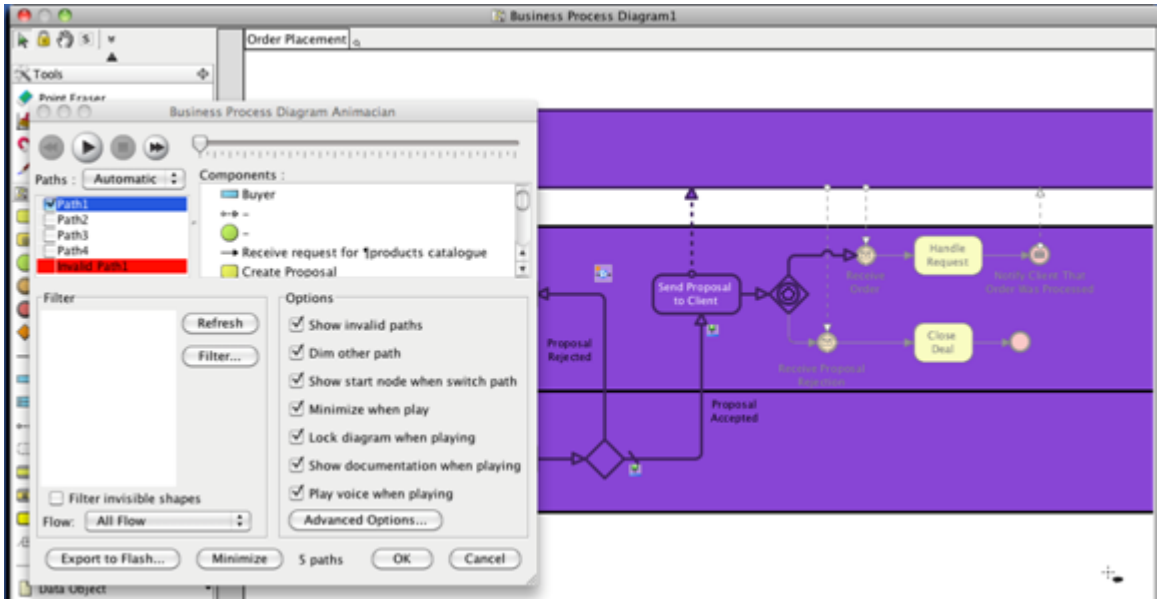

Рисунок 16. Animatian

Simulatian – еще более интересная опция, с помощью которой можно моделировать бизнес-процессы и наблюдать за их течением.

К примеру, известно, что работу А выполняют два человека за 3 часа, после чего задача/предмет, над которым происходит действие, переходит к другому специалисту, которому необходимо 20 минут для выполнения работы над некой сущностью. Весь этот процесс можно смоделировать, указав количество сущностей над которыми необходимо проделать действия и количество ресурсов, выполняющих действия над сущностями. Таким образом, можно смоделировать выполнения процесса в реальном времени и выявить его недостатки. (см. рисунок 17)

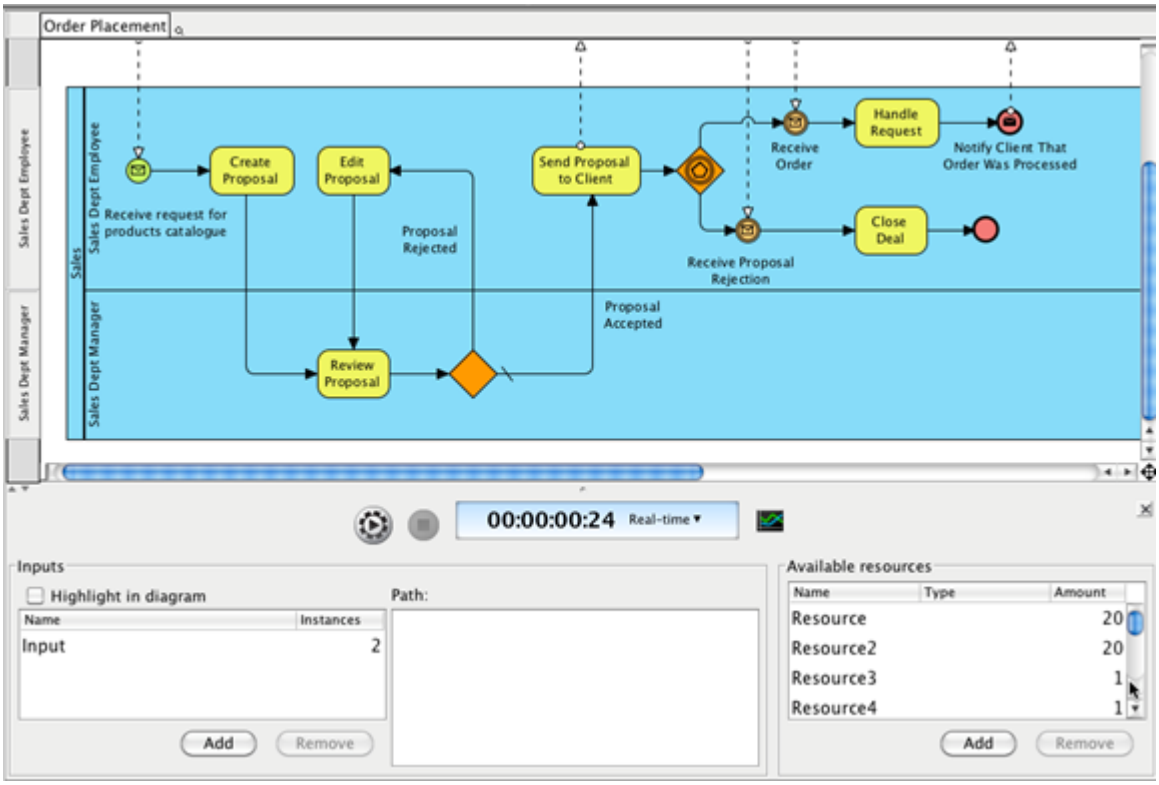

Рисунок 17. Simulatian

В дополнение к описанному выше, далее приведен сравнительный анализ менее существенных положительных и отрицательных сторон VP for UML:

Положительные стороны:

1. Для каждого элемента диаграммы можно записать аудиофайл в дополнение к документации.
2. Visual Paradigm Suite разработан на Java и является кроссплатформенной системой.
3. Большое количество обучающих материалов на сайте компании и возможность записаться на платные семинары.
4. Наличие глоссария и постоянно-работающей проверки орфографии.
5. Возможность проектирования баз данных, генерации кода запросов и хранимых процедур.

Отрицательные стороны:

1. Для Visual Paradigm Suite существует целый набор различных лицензий и, для того, чтобы определиться, какая конкретно подойдет, пользователю придется внимательно прочитать 20 страниц текста.

2. Отсутствие возможности открыть проект, созданный в более новой версии, даже если обновление системы произошло с версии 4.1 до 4.2 и никаких принципиальных изменений между версиями не наблюдается.
3. Экспорт диаграммы в изображение потребляет большое количество ресурсов системы.
4. При передвижении элементов в BPD-диаграммах наблюдалась задержка между реальным движением мыши и смещением элемента диаграммы.

В целом программа Visual Paradigm Suite 5.0 – достаточно неплохой продукт для управления требованиями и построения визуальных моделей, способный составить конкуренцию даже самым передовым и популярным инструментам для UML-моделирования. [6]

Rational Software Architect

IBM Rational Software Architect Designer (RSAD, ранее - RSA) - это комплексное средство проектирования, моделирования и разработки программного обеспечения. Rational Software Architect Designer строится на основе среды программного обеспечения с открытым исходным кодом Eclipse и расширяется с помощью множества подключаемых модулей Eclipse. Если требуется выполнить какие-либо особые требования, можно отдельно приобрести соответствующие расширения Rational.

Rational Software Architect Designer помогает лучше управлять архитектурой и получать эффективные результаты за счет следующих преимуществ:

1. **Поддержка моделирования на базе UML** и инструменты разработки на основе моделей (MDD) помогают рационализировать создание приложений и служб Java и Web 2.0.
2. **Мощные инструменты и средства управления процессами** помогает упростить разработку и повысить ее качество и эффективность.
3. **Доступ к облачным службам** дает преимущества масштабируемых служб инфраструктуры.
4. **Гибкая расширяемая платформа** помогает доставлять высококачественное программное обеспечение с более быстрой окупаемостью (ROI).

Rational Software Architect версии 7.5 включает в себя возможность прямых преобразований:

1. UML в Java

2. UML в C#
3. UML в C++
4. UML в EJB
5. UML в WSDL
6. UML в XSD
7. UML в CORBA IDL
8. UML в SQL на основе логической модели данных, поддерживаемой программным обеспечением IBM Rational(сейчас называется Rational Software).

В тоже время программа предоставляет возможность обратных преобразований Java в UML, C++ в UML, .NET в UML.

Благодаря использованию Eclipse в качестве фундамента вы можете просто и быстро наращивать объем функциональных возможностей Rational Software Architect в соответствии с конкретными требованиями проекта. Eclipse также поддерживает "экосистему" плагинов независимых разработчиков, которые еще больше расширят ваши возможности по созданию оптимальной среды разработки приложений. А поскольку платформа Eclipse написана на Java, вы можете экипировать свою команду инструментами разработки на основе моделей как в среде Windows, так и в среде Linux. Основанный на технологии Eclipse, Rational Software Architect предоставляет пользователям открытый, в высшей степени расширяемый и настраиваемый инструмент, поддерживающий разработки в масштабе предприятия.

В программном продукте Rational Software Architect использовуются новейшие достижения в технологии языков моделирования.

Rational Software Architect поддерживает UML версии 2 (UML 2), включая структурированные классы и усовершенствования, внесенные в циклограммы, диаграммы действий и диаграммы конечных автоматов. Благодаря этим и другим доработкам стандарта пользователи могут описывать свою архитектуру с большей степенью четкости и контроля, чем раньше. Группа Object Management Group (OMG) вывела "выразительность" языка в части управления процессом на новый уровень, выработав инициативу Model Driven Architecture (MDA). Rational Software Architect поддерживает MDA, позволяя пользователю определять несколько уровней моделей, связанных с определенными пользователем преобразованиями между моделями и кодом, результатом чего является более четкое разделение аспектов жизненного цикла. На рисунке 18, 19 представлены примеры диаграмм.

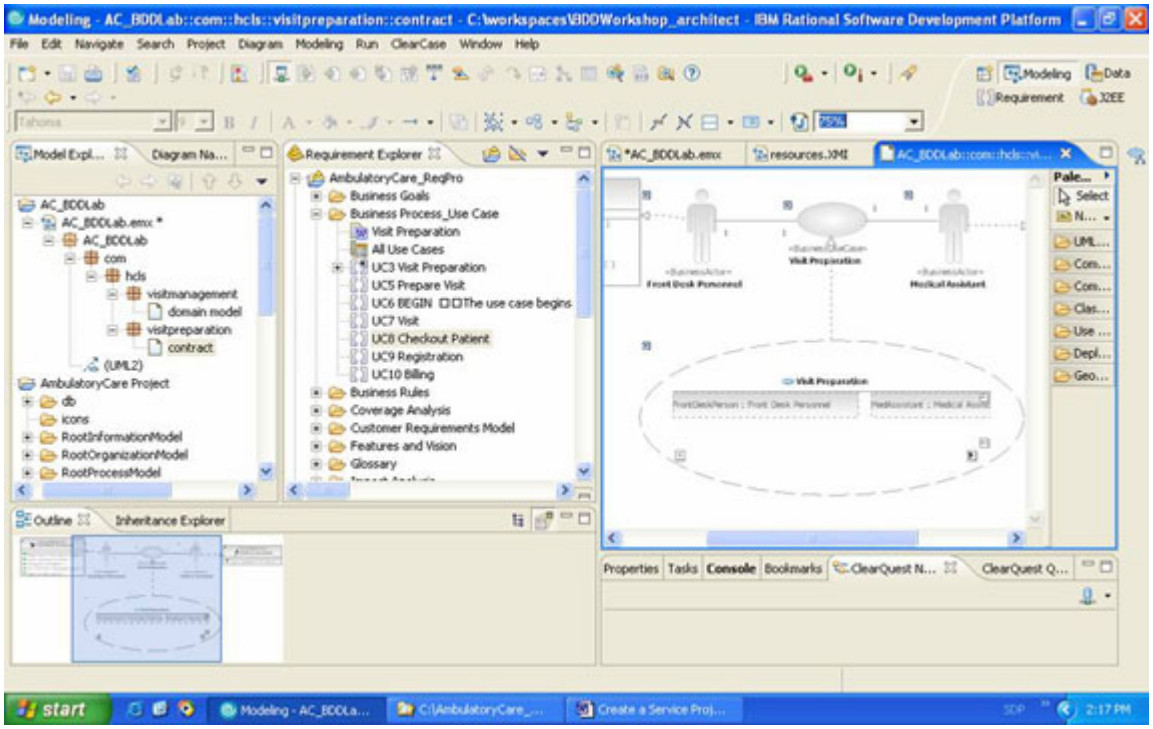

Рисунок 18. Диаграмма вариантов использования в программе Rational Software Architect

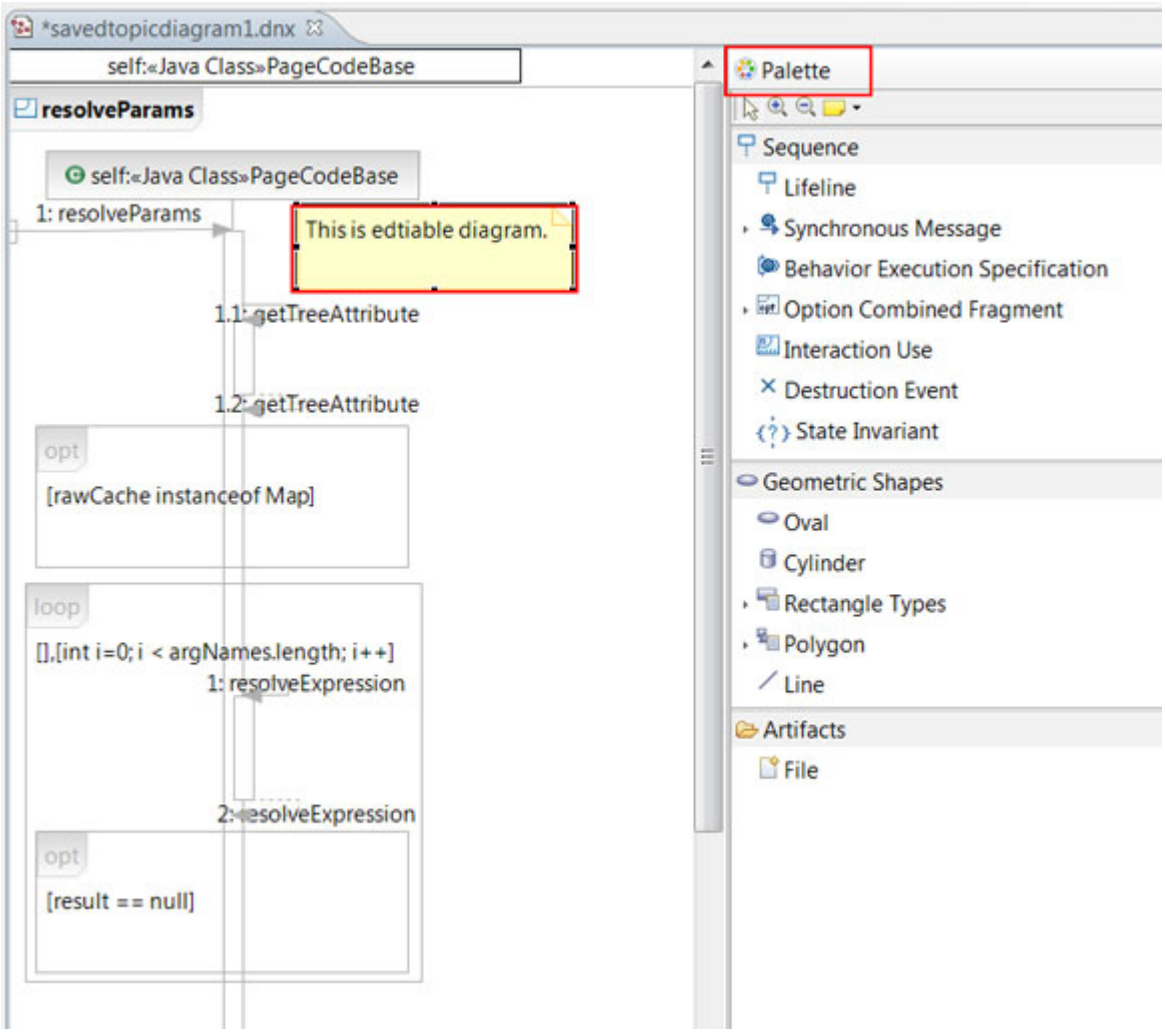

Рисунок 19 – Элементы диаграммы последовательности

Rational Software Architect упрощает среду разработки и проектирования. Поскольку продукт включает в себя все возможности Rational Application Developer for WebSphere Software, полной среды IDE корпоративного класса, специалисты получают в свое распоряжение полностью интегрированную среду проектирования и разработки в одном дистрибутиве, устанавливаемую одновременно. Это дает пользователям единый инструмент для разработки и проектирования, упрощает оценку, приобретение и интеграцию этих компонентов среды разработки ПО.

В программе возможна интеграция с другими аспектами управления жизненным циклом разработки. Rational Software Architect помогает специалистам осуществить интеграцию с другими аспектами управления жизненным циклом. Можно обращаться к требованиям ТЗ, хранящимся в Rational RequisitePro, выбирать из них те, которые связаны с соответствующими элементами моделирования, и выполнять синхронизацию по выбранным пользователем правилам. Пользователи могут генерировать отчеты, освещающие контролируемость связей между требованиями

ТЗ и проектированием. Файлами моделирования можно управлять с помощью Rational ClearCase LT, мощного продукта для управления конфигурацией, который поставляется вместе с Rational Software Architect. В качестве альтернативного варианта пользователи, работающие с инструментом Concurrent Versions System (CVS), могут интегрировать продукт с этой системой. А интеграция с IBM Rational Unified Process (RUP) предоставляет группам разработчикам возможность общего, интерактивного и интегрированного управления процессом.

Программный продукт Rational Software Architect предоставляет возможность разрабатывать такие диаграммы как:

1. Диаграмма активности;
2. Диаграмма классов;
3. Диаграмма связей;
4. Диаграмма компонентов;
5. Диаграмма составных структур;
6. Диаграмма развертывания;
7. Диаграмма взаимодействий;
8. Диаграмма пакетов;
9. Диаграмма машин состояния;
10. Диаграмма синхронизации;
11. Диаграмма прецедентов и др.

Программный продукт оснащен большим функционалом и подходит для проектирования информационных систем.

ЗАКЛЮЧЕНИЕ

В данной работе были представлены материалы об объектно-ориентированном подходе при проектировании информационных систем. Базовым средством документирования результатов проектирования информационных систем является язык UML.

В настоящее время существует достаточно много программных средств, автоматизирующих процесс проектирования информационных систем. Так как проектирование является важным этапом жизненного цикла разработки программного обеспечения, ему уделяется особое внимание.

В работе были рассмотрены такие программные средства как Sparx Systems Enterprise Architect, Visual Paradigm, Rational Software Architect. Данные продукты были выбраны для анализа, так как они часто обсуждаются пользователями и являются наиболее популярными. В работе был рассмотрен функционал каждой из систем.

Программное обеспечение, автоматизирующее процесс проектирования, делает работу системных аналитиков, IT-специалистов проще, это предоставляет возможность сотрудникам говорить на одном языке, используя единые исходные материалы, нотацию и т.д.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Попов А.В., Григорьева А. Л., Лошманов А. Ю. Объектно-ориентированный анализ, проектирование и программирование информационной системы университета // Современные проблемы науки и образования. – 2012 (дата обращения: 09.11.2017)
2. Официальный сайт НОУ ИНТУИН - [Электронный ресурс] - http://www.intuit.ru/studies/courses/10/10/lecture/298?page=2 (дата обращения: 08.11.2017).
3. Курс лекций по проектированию информационных систем - [Электронный ресурс] - https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema9 (дата обращения: 10.11.2017).
4. Учебные материалы по языку проектирования UML - [Электронный ресурс] – http://book.uml3.ru/sec_1_5 (дата обращения: 08.11.2017).
5. Официальный сайт компании Sparx Systems - [Электронный ресурс] – http://www.sparxsystems.com/ (дата обращения: 09.11.2017).
6. Материалы по продукту Visual Paradigm - [Электронный ресурс] - http://analyst.by/articles/vpforuml8 (дата обращения: 10.11.2017).