top
КОМПЬЮТЕРНАЯ АКАДЕМИЯ

{ПРОГРАММИРОВАНИЕ}

UNITIED MODELING LANGUAGE

Урок №4

Диаграмма последовательности, диаграмма кооперации, диаграмма компонентов и диаграмма развертывания

Содержание

1. Диаграмма последовательности

1.1. Цели данного типа диаграмм

Моделирование системы имеет два основных аспекта: моделирование структуры системы (набора составляющих её компонентных элементов) и определение связей между компонентами системы.

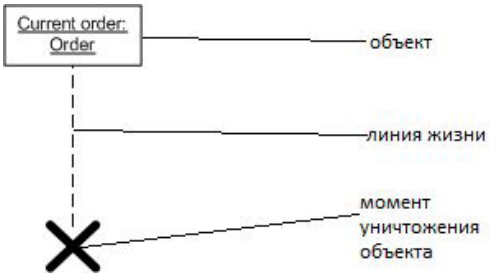
Говоря связи, мы понимаем, что компоненты системы некоторым образом друг с другом взаимодействуют. Один из основных принципов объектно-ориентированного анализа и проектирования гласит, что всякие способы информационного обмена между элементами системы должны выражаться в отправке и получении ими сообщений друг от друга.

Для представления специфики взаимодействия компонентов системы друг с другом в UML используются два типа диаграмм, о которых пойдёт речь в настоящем уроке. Это диаграммы последовательности и диаграммы кооперации. И если диаграммы кооперации отражают структурную сторону взаимодействия элементов системы, то диаграммы последовательности призваны иллюстрировать временную сторону этого взаимодействия, в чём и состоит основная цель их использования.

1.2. Базовые понятия

1.2.1. Объекты

Для определения составных элементов системы мы до сих пор использовали термины компонент и элемент. Но, когда речь заходит о реальном взаимодействии, то мы, как правило, уже имеем действие с реальными объектами (objects) (будь то программный объект – действующий экземпляр некоторого класса или объект предметной действительности). Поэтому диаграммах последовательности объект является ключевым понятием описывающим источник активности.

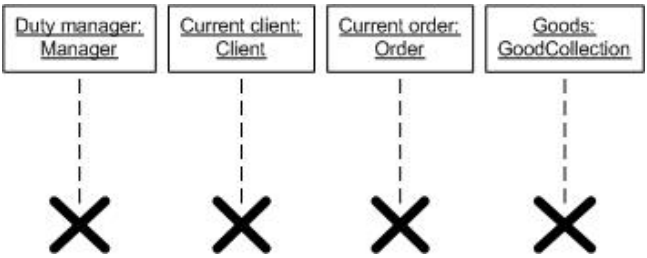


Взаимодействие подразумевает наличие нескольких участников, поэтому на диаграммах последовательности мы будем иметь дело с несколькими объектами. Стоит упомянуть, что диаграмма последовательности содержит только те объекты, которые участвуют в описываемом взаимодействии.

Объект на диаграммах последовательности изображается в виде прямоугольника расположенного в начале своей линии жизни (понятие линии жизни будет рассмотрено дальше в соответствующем разделе), с написанным в нём

именем объекта и классом, к которому объект принадлежит. Имя объекта от имени класса отделяется двоеточием, как это показано на приведённом слева изображении. Однако, там, где указывать имя класса не обязательно, можно указать только имя объекта.

Объекты на диаграмме должны располагаться слева на право таким образом, чтобы крайним слева был тот объект, который инициирует взаимодействие.



Временная шкала в диаграмме направлена сверху вниз. Поэтому начальному моменту взаимодействия соответствует верхняя часть диаграммы.

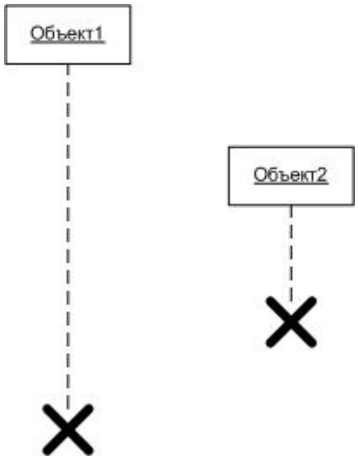
1.3. Линия жизни объекта

Линия жизни объекта (object lifeline) используется для того, чтобы продемонстрировать, какой период времени объект существует в системе и, соответственно, может участвовать во взаимодействии.

Линия жизни, как уже было показано выше, изображается в виде вертикальной пунктирной линии,

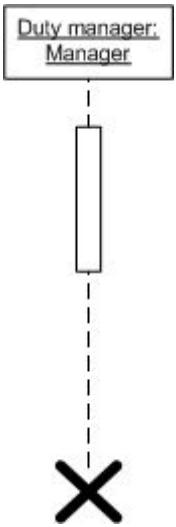
направленной от обозначения объекта сверху вниз до момента «уничтожения» объекта, который также можно назвать точкой завершения линии жизни. Момент уничтожения объекта изображается в виде латинской буквы «X», размещённой в нижней части линии жизни объекта.

Необходимо помнить, что далеко не все объекты существуют с начала осуществления описываемого в диаграмме взаимодействия. Некоторые объекты могу использоваться как вспомогательные. Они создаются по мере необходимости и уничтожаются, как только необходимость в их существовании пропадает. Подобное использование объектов имеющих ситуативное значение, как правило, связано с рациональным расходом ресурсов системы с целью повышения её эффективности. Временные объекты изображаются таким образом, чтобы прямоугольник, обозначающий объект, находился не в верхней части диаграммы, а смещённым по вертикали к тому месту, которое соответствует моменту его создания, как это показано на приведённом слева изображении.

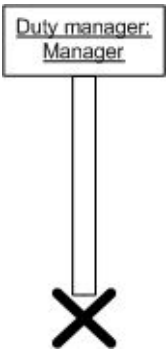


1.4. Фокус управления

Фокус управления (focus of control) – это графическое отображение активного состояния объекта в пределах его линии жизни на диаграмме.



На протяжении всей линии жизни объекта его активное состояние может сменяться пассивным. Под активностью понимается выполнение объектом некоторых действий. Однако, далеко не каждый объект на протяжении всей своей линии жизни постоянно находится в активном состоянии. Поэтому в языке моделирования UML водиться понятие фокуса управления, которое графически изображается в виде вытянутого прямоугольника, расположенного на линии жизни объекта, и соответствующего активному состоянию объекта. Под пунктиром линии жизни, в свою очередь, понимается состояние ожидания объектом сообщений, инициирующих его активность.



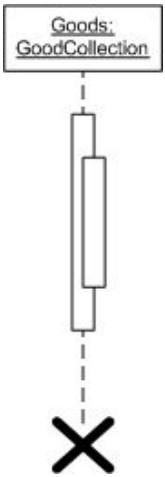
На предложенном справа изображении представлено графическое представление активного состояния на линии жизни объекта.

Бывают ситуации, когда объект находится в активно состоянии на всём протяжении своего существования в системе. В таких случаях его линия жизни заменяется изображением фокуса управления, как это показано на изображении слева.

Бывают также ситуации, в которых источником активности выступает пользователь или некоторое стороннее по отношению к системе «действующее лицо», тогда в качестве инициатора активности (крайнего левого объекта) изображают «действующее лицо» (actor). При этом линия жизни

«действующего лица» не имеет символа уничтожения, что подчёркивает характерную особенность пользователя присутствовать на всём периоде активности системы.

Действующее лицо может иметь имя, а может выступать в виде анонима. Пример графического изображения анонимного действующего лица приведён на изображении справа.



В диаграммах последовательности существует специальное обозначение для изображения рекурсивного вызова. В таких случаях принято говорить, что объект инициирует взаимодействие с самим собой. Рекурсия изображается в виде вертикально вытянутого прямоугольника прикреплённого с правой стороны фокуса управления объекта, осуществляющего рекурсивное взаимодействие.

Справа приведён пример графического изображения рекурсивного взаимодействия.

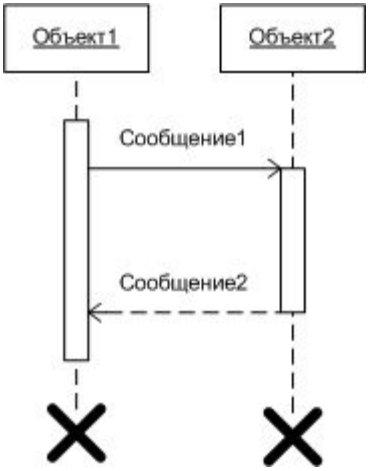
Существуют также ситуации, при которых объект инициирует обращение к самому себе не рекурсивного характера (например, это может интерпретироваться на уровне спецификации как вызов объектом собственного метода). Тогда говорят о рефлексивном сообщении, которое графически изображается, как стрелка направленная из некоторого фокуса управления в него же. Пример рефлексивного сообщения представлен на изображении справа.

1.5. Сообщения

Как упоминалось в начале раздела, ель диаграмм последовательности состоит в том, чтобы отразить временной аспект взаимодействия объектов системы. Основным направлением в рамках этой цели является изображение самого взаимодействия.

Нами оговаривалось, что согласно принципам объектно-ориентированного анализа и проектирования, всякое взаимодействие объектов некоторой системы может осуществляться лишь посредством отправки сообщений между объектами. В таком контексте, сообщение (message) – это законченный фрагмент информации, который один объект отправляет другому. Также предполагается, сто факт приёма объектом сообщения инициирует некоторые действия, которые должны быть выполнены объектом, получившим сообщение.

Обычно говорят о том, что сообщение инициирует выполнение некоторых операций, а аргументы (параметры) этих операций передаются в «теле» сообщения. При этом сообщения должны быть упорядочены по времени их возникновения в системе.



Графически сообщения на диаграммах последовательности изображаются в виде горизонтальных стрелок, соединяющих линии жизни или фокусы управления двух объектов. Стрелки сообщений должны

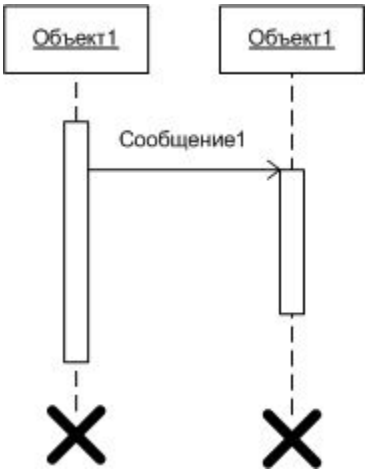
быть направленны от объектов, отправляющих сообщения (их называют клиентами), к объектам, которые их получают (такие объекты называют серверами). Принято считать, что сообщение имеет форму запроса, а реакцией сервера на его выполнение служит «ответ», то есть обратное сообщение клиенту о том, что запрашиваемые действия были выполнены. Ответ передаётся тоже в форме сообщения. Ответ обычно изображается в виде горизонтальной пунктирной стрелки, направленной от сервера к клиенту.

Принимается допущение о том, что время передачи сообщения относительно мало в сравнении с временем, которое тратиться на исполнение объектом-сервером запрашиваемого действия.

Также, принято считать, что за время передачи сообщения с соответствующим объектом не может произойти никаких изменений.

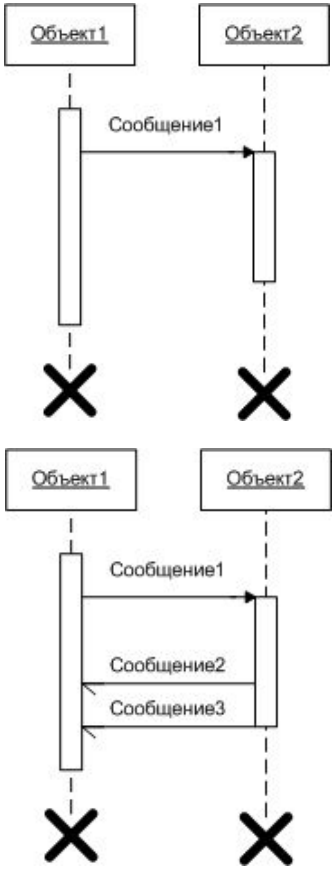
В языке моделирования UML встречаются различные виды сообщений, для каждого из которых определено своё графическое обозначение (справой стороны будет приведено графическое обозначение для каждого типа соединений):

- синхронное сообщение – наиболее часто встречаемый вид сообщений. Он используется для вызова процедур, выполнения операций и обозначения вложенных потоков управ-



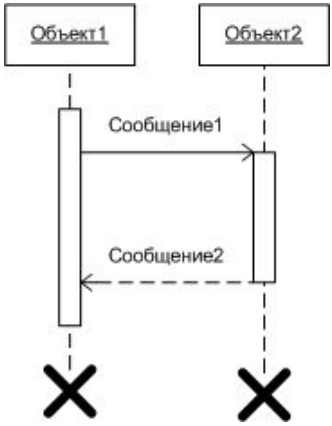
ления. Начало такой стрелки всегда соприкасается с линией жизни или фокусом управления объекта-клиента, а конец – с линией жизни объекта-сервера. При этом типе соединения принимающий объект обычно получает и фокус управления, становясь активным.

- сообщение вызова – используется для обозначения невложенного потока управления. Такая стрелка обычно изображается направленной от линии жизни (фокуса управления) одного объекта к фокусу управления объекта-сервера и указывает на один шаг потока управления. Подобные вызовы, как правило, асинхронны.



- асинхронное сообщение – определяет асинхронное сообщение между двумя объектами в некоторой последовательности. Примером в чистом виде асинхронного сообщения является отправка объектом-сервером сообщений о том, что процедура переходит на следующий этап. Или же возврат сообщения о том, что произошла некоторая ошибка.

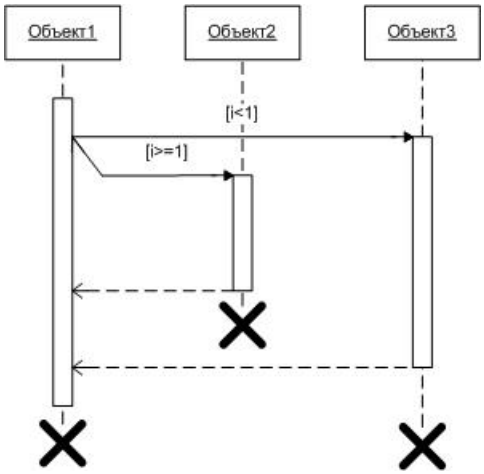
- сообщение возврата – определяет, что сервер по выполнении действия вернёт в вызывающий объект некоторый результат. Сообщение возврата графически изображается горизонтальной пунктирной стрелкой направленной от объекта-сервера к объекту-клиенту. В процедурных потоках управления сообщение возврата может быть опущено в силу того, что из контекста становиться понятно, что сервер должен возвращать результат. Также принято считать, что каждый вызов процедуры имеет парный ему возврат.



Для асинхронных вызовов стрелка возврата должна быть указана явно, поскольку из контекста не известно, предусматривает ли каждый конкретный асинхронный вызов возврат результата.

1.6. Ветвление потока управления

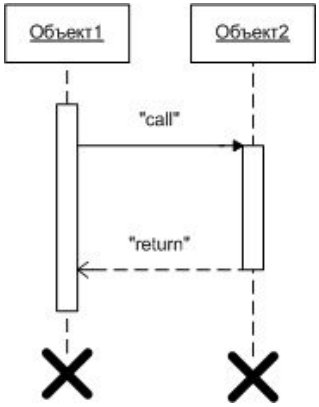
Ветвление потока управление графически изображается как несколько стрелок выходящих из одного фокуса управления клиентского объекта к нескольким объектам-серверам. При этом возле каждой стрелки в квадратных скобках указывается условие отправки соответствующего сообщения. Предполагается, что условия, используемые в ветвлении, должны взаимно исключать друг друга. Ниже приведён пример графического изображения ветвления в диаграммах последовательности языка моделирования UML.



1.7. Стереотипы сообщений

Стереотип сообщения – это некоторое стандартное действие, которое выполняется в ответ на сообщение. Стереотип может быть указан в двойных кавычках возле сообщения (как правило, над соединением с выравниванием по центру).

Справа представлен пример графического оформления использования стереотипа сообщения.



В языке моделирования UML предусмотрен набор стандартных стереотипов:

- “call” (вызвать) – сообщение, предполагающее вызов процедуры объекта-сервера.
- “return” (возвратить) – сообщение, которое возвращает значение, завершённой процедуры.

- “create” (создать) – сообщение, которое требует создания нового объекта, для выполнения некоторых операций.
- “destroy” (уничтожить) – сообщение, которое требует уничтожения соответствующего объекта.
- “send” (отправить) – означает отправку некоторого сигнала принимающему объекту. Отличие сигнала от сообщения заключается в том, что сигнал должен быть явно определён в классе, объект которого является отправителем.

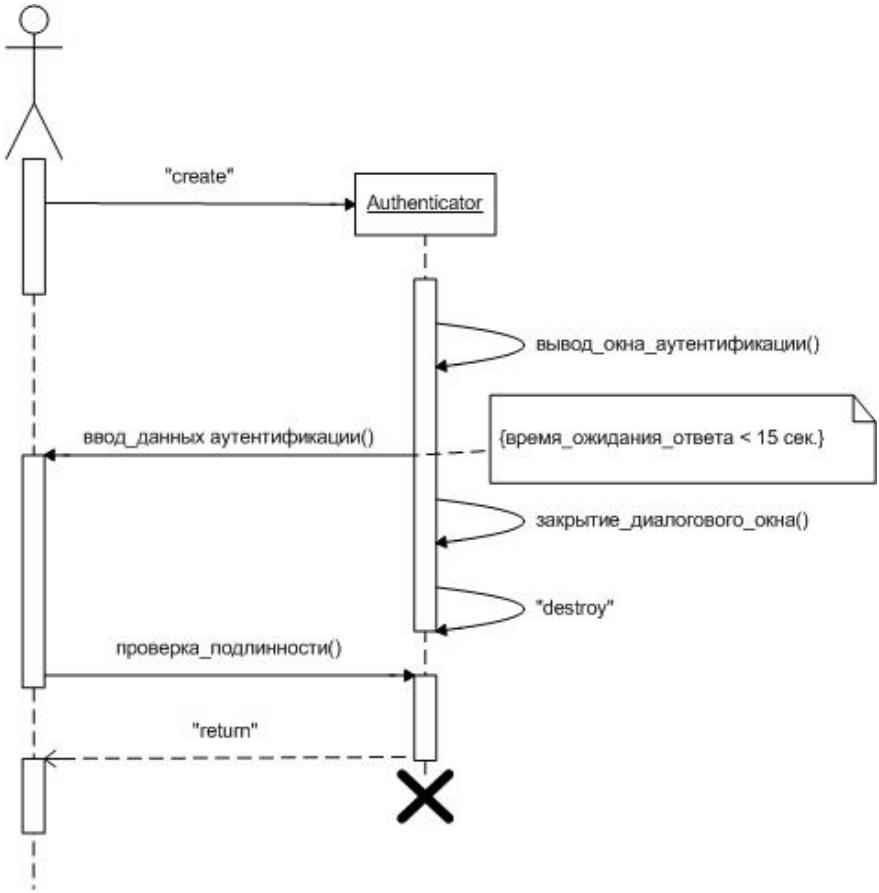
Сообщения могут иметь также и указания конкретных операций, которые они инициируют в принимающих объектах. Для спецификации вызова операции в принимающем объекте рядом со стрелкой сообщения необходимо указать имя операции, вызов которой осуществляется, и круглые скобки после неё, в которых можно указать аргументы этой операции.

Временные ограничения на диаграммах последовательности. Бывают такие случаи, что возникает необходимость указания ограничений по времени при отправке сообщения. Ограничения могут накладываться как на сам процесс передачи сообщения, так и на время выполнения вызываемой операции. Временное ограничение записывается в таком же контейнере, как и комментарии, (прямоугольник с заломленным верхним углом) с тем отличием, что для записи временных ограничений в диаграммах последовательности используют фигурные скобки. Временные ограничения могут быть указаны в начале стрелки сообщения, хотя, обычно, их указывают слева от стрелки соответствующего сообщения. Если временное

ограничение накладывается на конкретный объект, то сначала указывается имя объекта, затем ставиться точка, а потом специфицируется временное ограничение на объект.

Важно помнить, что временные ограничения – это не условия, они имеют характер директив.

Ниже представлен пример использования временных ограничений в диаграммах последовательности языка моделирования UML.



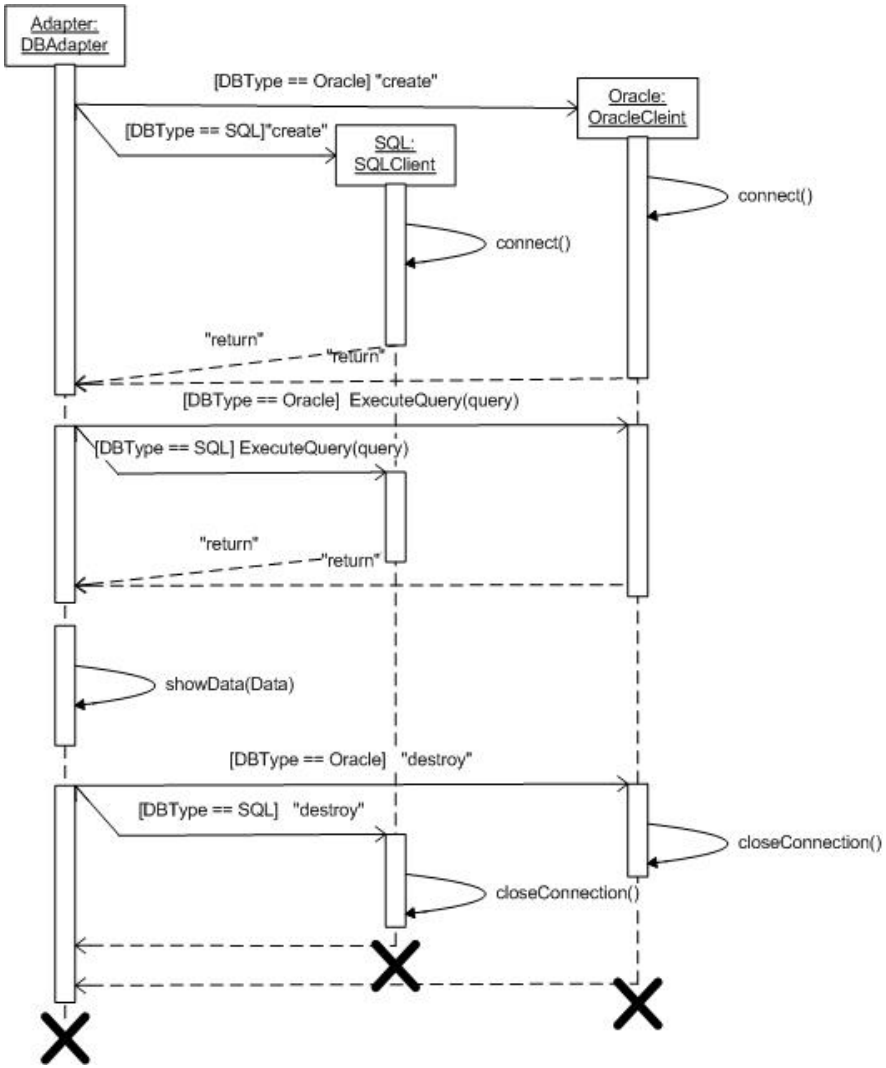
1.8. Комментарии или примечания

Комментарии (примечания важная особенность любого формального языка, в том числе и UML. Их важность состоит в том, что они позволяют детализировать и пояснить неочевидные и двусмысленные формулировки (в данном случае в моделях). Как уже упоминалось, комментарии указываются в прямоугольном контейнере с заломленным верхним правым углом. А от самого контейнера следует пунктирное линейное соединение к комментируемому формализму.

1.9. Практические примеры построения диаграмм последовательности

В качестве примера мы хотим привести пример построения диаграммы последовательности работы объекта класса DBAdapter, являющегося адаптером для подключения и выполнения запросов к серверам баз данных SQL Server и Oraclce.

Условием использования класса является задание влага, определяющего с каким СУБД будет осуществляться взаимодействие. На основании этого флага адаптер создаёт необходимый объект клиента баз данных и впоследствии использует его для выполнения запросов к необходимой базе данных. После того, как данные получены и выведены пользователю, адаптер уничтожает объект клиента, собственно, реализующий взаимодействие, который, в свою очередь, перед уничтожением закрывает соединение с СУБД.


Adapter:
DBAdapter
[DBType == Oracle] "create"
Oracle:
OracleCleint
[DBType == SQL]"create"
SQL:
SQLClient
connect()
connect()
"return"
"return"
[DBType == Oracle] ExecuteQuery(query)
[DBType == SQL] ExecuteQuery(query)
"return"
"return"
showData(Data)
[DBType == Oracle] "destroy"
[DBType == SQL] "destroy"
closeConnection()
closeConnection()

2. Диаграмма кооперации

2.1. Цели данного типа диаграмм

Как и диаграммы последовательности, диаграммы кооперации используются для изображения особенностей взаимодействия объектов моделируемой системы. Но, в отличие от диаграмм последовательности, диаграммы кооперацийпредназначены для изображения структурного аспекта описываемого в них взаимодействия.

Диаграммы кооперации отражают только отношения между объектами, имеющие определённый смысл с точки зрения взаимодействия, а не их последовательность. С другой стороны последовательность вызовов на диаграммах кооперации может быть отражена при помощи специфицирования порядковых номеров объектам, но этот способ, всё равно, в полной мере не отражает временной аспект взаимодействия. Поэтому важно понимать, что для комплексного отражения модели не достаточно использовать какой-то один способ представления.

Однако, временной аспект важен не на всех уровнях абстракции. Например, с точки зрения аналитики или архитектуры важно скорее структурное описание модели, отражающее всё богатство составляющих систему объектов и существующих между ними связей. Такое структурное представление и является целью использования диаграмм кооперации.

2.2. Базовые понятия

Кооперация

Кооперация (collaboration) – формализм языка моделирования UML, который служит для отображения множества объектов, взаимодействующих с определённой целью. Цель кооперации состоит в том, чтобы проиллюстрировать особенности основных операций системы (тех операций, которые имеют наибольшее значение с «точки зрения системы»).

Кооперация имеет два уровня отображения:

- **уровень спецификации**, на котором отображаются роли классификаторов и ассоциаций, существующих в рамках определённого взаимодействия;
- **уровень примеров** иллюстрирует экземпляры (объекты) и связи, которые образуют «роли» в рамках кооперации.

Диаграмма кооперации уровня спецификации отражает роли, исполняемые участвующими во взаимодействии элементами. Под элементами на этом уровне понимаются классы и ассоциации

В диаграмме уровня примеров в качестве действующих лиц уже выступают не классы и ассоциации, а их экземпляры, то есть объекты и связи (связи на данном уровне дополнены стрелками сообщений, указывающими направление передачи сообщения).

2.3. Диаграмма кооперации уровня спецификации

Кооперация уровня спецификации изображается в виду пунктирного эллипса, с указанным внутри него именем

кооперации. Важно помнить, что кооперация уровня спецификации необходимо связана с конкретным вариантом использования и призван описать детали его реализации, поскольку, в зависимости от различных вариантов использования, особенности кооперации могут резко отличаться.

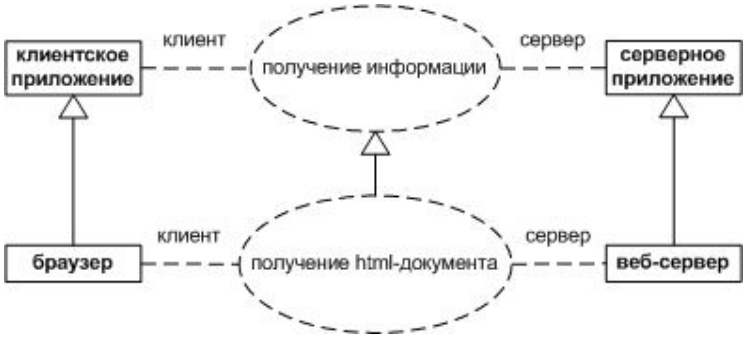
Участники кооперации соединяются с ней пунктирными линиями. В качестве участников кооперации могут выступать объекты или классы. При этом возле каждой соединительной линии необходимо указать роль (role) участника в рамках кооперации (роли, как правило, соответствуют именам элементов в контексте взаимодействия).

На изображении сверху показано, как выглядят кооперации и соединения графически на диаграммах кооперации.

Класс, как видно из приведённого рисунка, графически изображается в виде прямоугольника с записанной в него строкой текста, которая называется ролью классификатора (classifier role). Роль классификатора призвана показать особенность использования объекта во взаимодействии. Роль классификатора, как правило, содержит только указание имени класса, хотя возможно также указывать и секции атрибутов и операций. Имя роли классификатора должно указываться в следующем формате:

```
имя_роли_классификатора: имя_классификатора
```

Имя классификатора – это, обычно, имя класса (типа данных), но также возможно указать полный путь к имени с полной спецификацией всех пакетов (если пакеты указываются, то необходимо специфицировать все пакеты в указанном пути). При этом имена пакетов отделяются друг от друга двойным двоеточием («::»).



Если кооперация подразумевает обобщённое представление, то есть одна кооперация является частным случаем другой кооперации, то на диаграмме могут указываться отношения обобщения между соответствующими кооперациями. Для этого используют обычное отношение обобщения, которое изображается сплошной линией, с не закрашенной стрелкой на конце. Отношение обобщения должно быть направлено от кооперации-потомка к кооперации-предку, как это показано на представленном выше примере.

Иногда возникает необходимость явно специфицировать, что кооперация является реализацией некоторой операции или классификатора. В таком случае

используют два способа графического отображения: первый способ заключается в том, чтобы соединить кооперацию с классом пунктирной линией, которая заканчивается стрелкой обобщения; второй способ заключается в том, чтобы указать через две точки после имени кооперации спецификацию того, реализацией какой операции и какого класса является данная кооперация. Спецификация операции указывается в следующем формате:

```
имя_кооперации: имя_классификатора::имя_операции_
классификатора
```

Описание кооперацій начинается с уровня спецификации, на котором собственно и указываются все обобщения и реализации. После этого каждая кооперация должна быть описана на уровне примеров, для того, чтобы раскрыть особенности внутренней структуры взаимодействия с описанием характера связей и роле, в которых выступают объекты вв рамках взаимодействия.

На диаграмме кооперации уровня спецификации могут присутствовать именованные классы, с указанием роли класса, а так же анонимные классы, только с указанием роли.

2.4. Объекты

Напомним, что под объектом (object) понимается отдельный экземпляр класса, который создаётся на этапе исполнения программы.

Он наделён собственным именем и конкретными значениями атрибутов.

Графически объект изображается в виде прямоугольника, содержащего подчёркнутый текст (именно подчёркивание даёт возможность быстро визуально отличить объекты от классов) записанный в следующем формате:

```
имя_объекта/имя_роли_классификатора:имя_классификатора
```

Имя роли классификатора может быть не указано (в таком случае его необходимо опустить вместе со следующим за ним двоеточием).

Имя роли опускается в том случае, если существует всего одна роль, в которой могут выступать объекты данного классификатора, а значит, достаточно будет указать либо имя классификатора, либо имя роли. Далее мы перечислим возможные варианты, различных сигнатур имени объекта:

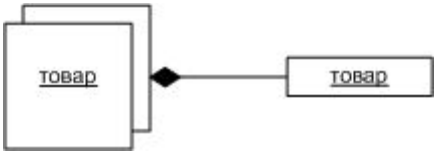
Шаблоны имён	Описание
:A	Анонимный объект – экземпляр класса A
/Role	Анонимный объект, играющий роль Role
/Role:A	Анонимный объект – экземпляр класса A, играющий роль Role
A/Role	Объект и именем A, играющий роль Role
A:C	Объект с именем A – экземпляр класса C
A/Role:B	Объект с именем A, который является экземпляром класса B и играет роль Role
A	Объект с именем A
A:	Объект неопределённого класса с именем A

Справа приведён пример того, как будут выглядеть объекты с различной сигнатурой имени на диаграмме.

2.5. Мультиобъект

Мультиобъект (multiobject) – это способ выразить множество объектов на одном из концов ассоциации в рамках диаграммы кооперации. Мультиобъект используют для того, чтобы показать, что сигнал отправляется сразу всем элементам некоторого множества объектов.

Графически мультиобъект изображается двумя прямоугольниками, наложенными друг на друга, при этом один из них выступает из-за верней правой вершины другого.



Также может быть явно специфицировано отношение композиции между объектом и мультиобъектом, в который он входит.

2.6. Активный объект

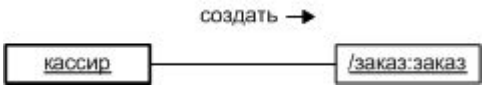
Все объекты можно условно разделить на два типа, которые в терминах языка моделирования UML носят название активных и пассивных.

Пассивные объекты выполняют операции только над данными и не могут управлять другими объектами. Однако, пассивные объекты, в рамках выполняемых ими запросов, имеют возможность отправлять сигналы другим объектам.

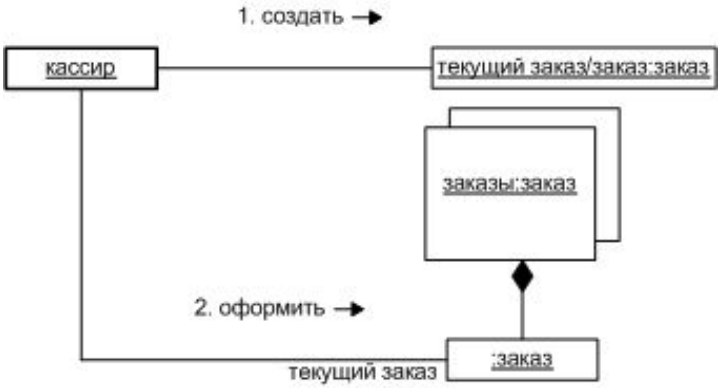
Каждый активный объект (active object) имеет нить или поток (thread) управления, и может управлять другими объектами. При этом предполагается, что поток управления может выполняться параллельно с другими потоками управления в рамках одного вычислительного процесса, который называют процессом управления (при этом понятие процесса и потока соотносимы с одноимёнными категориями операционной системы, и различаются по принципу выделения ресурсов; таким образом, под процессом управления можно понимать поток управления, который монополизирует ресурсы и включает в себя другие потоки управления).

Активные объекты изображаются как обычные объекты, прямоугольник которых имеет более широкие края. Также возможен вариант, при котором активный объект маркируется ключевым словом «{active}».

Активный объект может инициировать один поток управления и представляет собой исходную точку данного потока управления.



На рисунке слева показан пример, на котором объект кассир инициирует создание объекта класса заказ, выполняющего одноимённую роль.



Также потоки могут быть пронумерованы, чтобы указать, в какой последовательности они создаются.

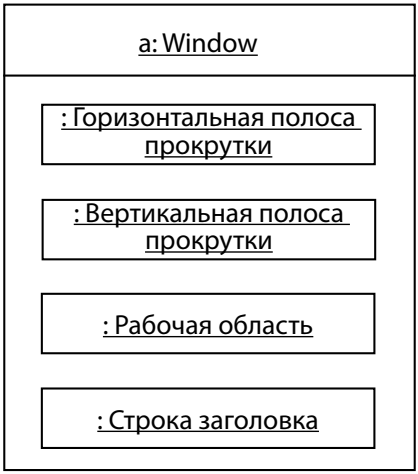
Справа представлен пример, на котором объект кассир создаёт объект «текущий заказ», который впоследствии оформляется, вызовом соответствующего действия. В данном случае на диаграмме указан только один активный объект – кассир.

2.7. Составной объект

Составной объект (composite object), который также называют объектом-контейнером, предназначен для графического изображения объекта, имеющего собственную структуру и собственные внутренние потоки управления. Составной объект – это экземпляр составного класса (или класса-контейнера), то есть класса, связанного отношениями агрегации или композиции со своими компонентными

частями. Очевидно, что соответствующие отношения связывают и экземпляры этих классов.

Графически составной объект изображается как обычный объект, вместо атрибутов которого указаны прямоугольники его компонентных классов.



Справа представлен пример, который иллюстрирует использование составного объекта. На примере показана структура объекта класса Window (то есть окно), которая имеет компонентные объекты, обеспечивающие соответствующий окну функционал (а именно: горизонтальную и вертикальную прокрутку, рабочую область и строку состояния).

2.8. Связи

Как уже упоминалось, ассоциации, как и классификаторы, тоже могут иметь экземпляр. Экземпляр ассоциации называется связью (link). Связь может существовать

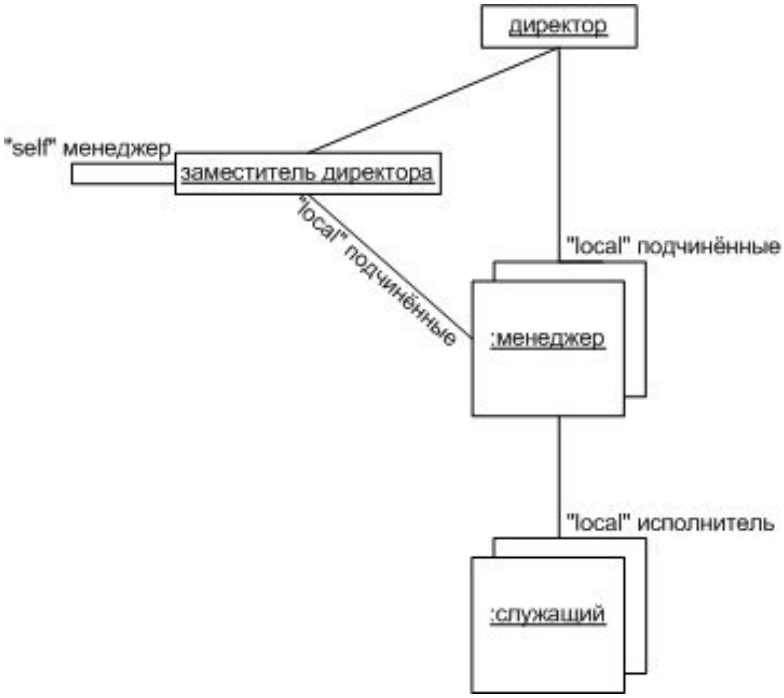
между двумя и более объектами. Бинарная связь (связь двух объектов) в диаграммах кооперации графически изображается в виде сплошной линии, проходящей от прямоугольника одного объекта к прямоугольнику другого. На обоих конах связи могут быть явно указаны роли соответствующих объектов, а рядом с линией, посередине, может быть указано имя данной ассоциации.

Связи не имеют имён, а также для них не указывается кратность. Хотя, для отдельных случаев ассоциации (таких как композиция и агрегация) могут быть указаны релевантные им обозначения.

2.9. Стереотипы связей

В языке UML предусмотрены стереотипы связей, которые указывают на особенность реализации связи, для которой они указаны. Ниже представлен список канонических стереотипов связей:

- “association” (ассоциация) – данный стереотип можно не указывать, поскольку по умолчанию предполагается, что связь является экземпляром ассоциации.
- “parameter” (параметр метода) – соответствующий данной связи объект может выступать только в роли параметра некоторого метода.
- “local” (локальна переменная метода) – область видимости переменной ограничена только соседним объектом.
- “global” (глобальная переменная) – область видимости соответствующей распространяется на всю диаграмме кооперации.



- “self” (рефлексивная связь объекта) – объект связан сам с собой. Такая связь допускает, что объект имеет возможность передавать сообщения самому себе. Графически данная связь изображается петлёй распложенной в верхней части объекта.

На приведённом справа изображении показан пример использования связей в диаграммах кооперации.

2.10. Сообщения

Сообщения уже рассматривались нами при изучении диаграмм последовательности. Однако, при построении диаграмм кооперации они имеют несколько дополнительных особенностей.

На диаграмме кооперации сообщения указывают, что некоторый объект передаёт другому объекту какую-то информацию. Другими словами при помощи сообщений мы указываем особенности коммуникации объектов нашей модели. При этом предполагается, что после получения сообщения объект выполнит некоторое действие (это принято называть стимулирующим смыслом сообщений).

Связь в таком контексте служит каналом для передачи сообщений от объекта-источника (source-object) к объекту-получателю, который также называют объектом назначения, (destination-object).

Графически сообщения изображаются помеченными стрелками рядом со связью (выше или ниже линии связи). Направление стрелки указывает от объекта-отправителя к объекту-получателю.

Существует четыре типа стрелок для обозначения сообщений:

1. → (сплошная линия с треугольной стрелкой) – используется тогда, когда сообщение инициирует вызов процедуры или другого потока исполнения. Как правило, все сообщения такого типа синхронны.
2. ↣ (сплошная линия с V-образной стрелкой) – пустой поток управления; изображает один отдельных «шаг» в последовательности потока управления. Чаще всего все такие сообщения асинхронны.
3. ⇀ (сплошная линия с полу-стрелкой) – используется когда, когда сообщение инициирует асинхронный поток управления. Сообщение данного типа отправляется в произвольный момент времени, который

изначально не известен, и чаще всего активными объектами. Обычно сообщения данного типа инициируются «актёрами» и являются стартовыми во всём потоке управления.

4. –⩥ (пунктирная линия с V-образной стрелкой) – используется для обозначения возврата из вызова процедуры. Как правило, сообщения такого типа отсутствуют на диаграммах вообще, поскольку по умолчанию предполагается наличие такого сообщения после окончания любой процедуры.

2.11. Формат записи сообщений

Сообщения могут иметь подпись в виде строки текста в следующем формате:

```
предшествующие_сообщения[сторожевое_условие] выражение_
последовательности
возвращаемое_значение имя_сообщения список_аургументов
```

- предшествующие сообщения выражаются в виде списка из разделённых запятыми номеров сообщений, указанных перед наклонной чертой – «/»;
- список предшествующих сообщений указывается с той целью, чтобы специфицировать, что данное сообщение не может быть передано до тех пор, пока не будут выполнены сообщения, указанные в списке.
- данное выражение может быть опущено. Но, в таком случае, оно опускается вместе с разделительной чертой;
- сторожевое условие – логическое выражение, которое указывается в квадратных скобках и предназначено

для того, чтобы синхронизировать отдельные потоки управления. Данное выражение может быть опущено;

- выражение последовательности – это список терминов (англ. term) последовательности, разделённый точками, после которого указано двоеточие;
- каждый из терминов представляет собой отдельный уровень вложенности последовательности процедур в форме законченной итерации. Первый термин соответствует началу процедурной последовательности.

Ниже представлена форма записи каждого отдельного термина:

```
[целое_число|имя][символ_рекурентности]
```

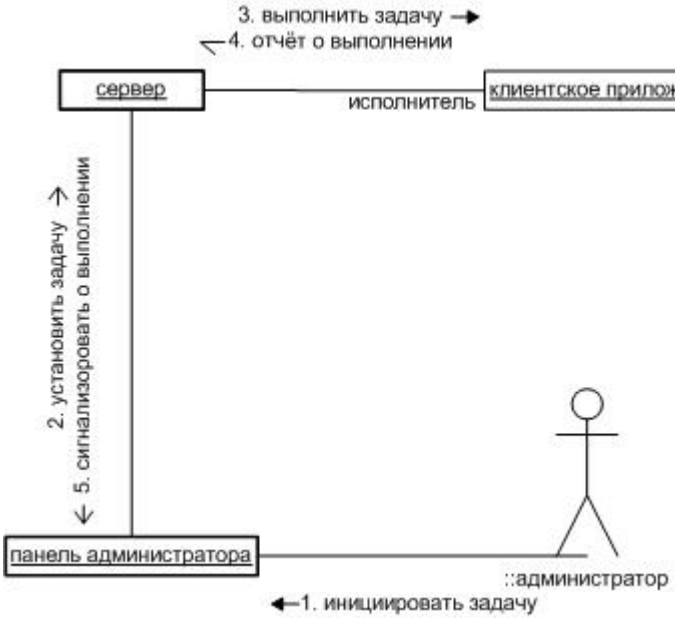
Целое число определяет порядковый номер термина в процедурной последовательности. Имя используется для того, чтобы указать параллельные потоки управления (сообщения, отличающиеся именем – параллельны на данном уровне вложенности). А символ рекурентности используется для указания условного или итеративного выполнения.

- возвращаемое значение – список имён значений, которые возвращаются по окончании коммуникации или взаимодействия в полной итерации указанной процедурной последовательности;
- имя сообщения – это имя события, которое инициируется в объекте получателе после того, как он его принял. Чаще всего в качестве имени указывают имя операции объекта-получателя;

- список аргументов – список действительных параметров вызываемой операции, разделённый запятыми и заключённый в круглые скобки.

2.12. Практические примеры построения диаграмм кооперации

В качестве примера составления диаграмм кооперации мы ходим привести составление общей структуры взаимодействия распределённого приложения, выполняющего удалённое администрирование рабочих станций. В качестве панели администратора выступает приложение, которое обеспечивает пользовательский интерфейс администрирования системы с возможностью установки задачи на выполнения всякому зарегистрированному клиентскому компьютеру.



После установки задания поступают на сервер, который в зависимости от того, когда рабочая станция будет доступной, отправляет клиентскому приложению сообщение о том, какое задание нужно выполнить.

После выполнения задания, клиентское приложение отчитывается серверу, а он, в свою очередь, сигнализирует панели администратора, чтобы представить администратору изменённую информаци.

3. Диаграмма компонентов

Диаграмма компонентов используется для отображения компонентов приложения. Построив такую диаграмму, вы сможете графически отобразить всю структуру приложения, начиная от объектов, классов, заканчивая исполняемыми файлами, и библиотеками.

Диаграммы компонентов состоят из компонентов, интерфейсов и связей между ними.

3.1. Цели данного типа диаграмм

Многократное использование кода – когда строится диаграмма компонентов, довольно легко заметить похожие части (компоненты) приложения, стоит задуматься имеется ли возможность из двух компонентов сделать один универсальный? Диаграмма компонентов поможет графически изобразить многократное использование кода.

Визуализация общей структуры программы – диаграмма компонентов очень хорошо отображает взаимосвязь объектов, это поможет программистам более детально разобраться со структурой объектов и их взаимосвязью.

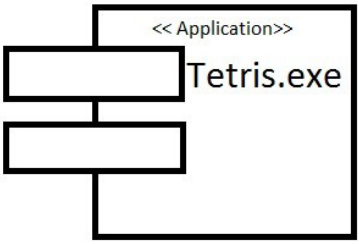
Представление структуры данных – с помощью диаграммы компонентов можно представить схему хранения данных, или схему работы любого приложения.

3.2. Базовые понятия

Как уже упоминалось, диаграммы компонентов состоят из трех базовых явлений:

- компоненты – основной строительный блок в диаграмме компонентов, с помощью компонентов обозначаются объекты классов, классы, функциональные точки, экземпляры программ.
- интерфейсы – диаграмма отображает интерфейсы доступа к компонентам, также на диаграмме можно обозначить ситуацию, когда для использования объекта вам необходим класс, поддерживающий определенный интерфейс.
- зависимости – с помощью зависимостей можно показать, что один объект использует другой. Зависимости иногда называют связью.

Компонент отображается на диаграмме компонентов как прямоугольник, в котором слева вставлены два прямоугольника:



Данный вид компонента не является стандартом, часто компонент изображен в виде простого прямоугольника, как правило, приложения, в которых составляются UML диаграммы, где-либо на компоненте рисуют данную фигуру, для обозначения, что данный прямоугольник является компонентом.

Для обозначения компонентов стоит использовать Microsoft Visual Studio 2010:

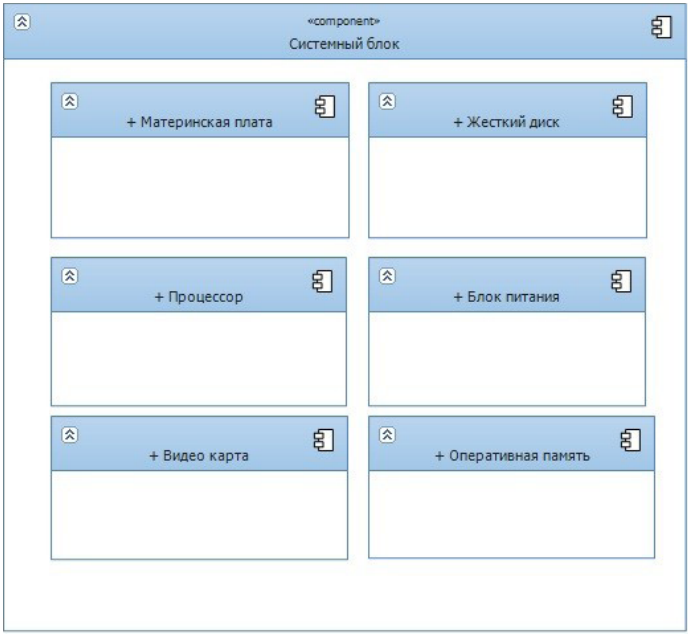
Как вы наверное заметили в правом верхнем углу прямоугольника изображена фигура указывающая на то что данный прямоугольник является компонентом. На компоненте, как правило, пишут две записи, это тип объекта, и его имя. Иногда вместо типа компонента ничего не пишут или пишут просто слово component. Тип объекта допустимо игнорировать, так как имеется стандарт написания имен, которые подразумевают тип объекта:

- скомпилированный код – когда компонент выступает, как готовое приложение возле его имени пишется расшерение испольняемого файла, например: Tetris.exe, или Tetris.jar такой же подход используется к библиотекам, например: MessageWindows.dll;
- файлы – когда приложение использует внешнее хранилище данных, например XML или TXT, имя такому компоненту дается с расширением файла. Например: Settings.xml;
- страницы – расширение для страниц на диаграмме компонентов тоже необходимо указывать. Например: Index.php;
- классы – когда речь идет о классах можно писать просто имя класса, но иногда пишут слово класс перед именем класса, например: class Program;

- объекты классов – Объекты именуется следующим образом: сначала пишется имя объекта, далее пишут двоеточье и пишут тип данных этого объекта, например: LWin:LoginWindow.

Рекомендуется указывать стереотип объекта, несмотря на то что его иногда игнорируют, это внесет ясность в вашу диаграмму.

Компоненты могут быть изображены на диаграмме по отдельности, а могут находиться друг в друге, например можно сказать что есть общее понятие системный блок компьютера, это компонент рабочей станции, но внутри системного блока находятся другие компоненты, такие как жесткий диск, материнская плата, процессор, блок питания. На диаграмме такая система отображается следующим образом:



Но эта диаграмма не является законченной, так как на ней отсутствуют связи и интерфейсы.

Стоит отметить, что вкладывать один компонент в другой можно только один раз, другими словами я не могу создать компонент системного блока, в котором будет находиться компонент оперативной памяти, внутри которой будут другие компоненты (чипы и т.д). Такое ограничение связанно с трудно читаемостью таких диаграмм.

Дизайнер в MS Visual Studio 2010 позволяет изменять основной цвет компонента, данное свойство имеет чисто эстетический характер.

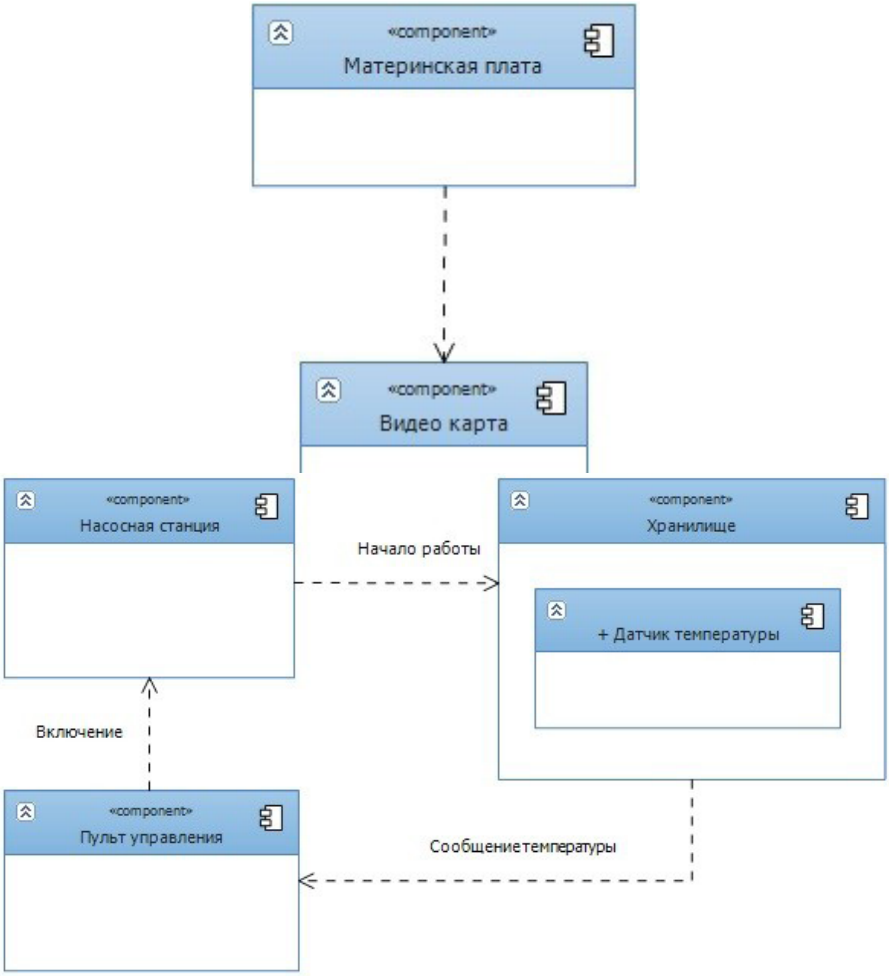
Зависимости – зависимость это простой способ показать на диаграмме что один компонент использует другой компонент, как правило, зависимость рисуется пунктирной стрелкой от зависимого объекта к не зависимому. Например: материнская плата в компьютере зависима от видео каты, а видео карта независимый компонент, потому стрелка должна указывать от материнской платы к видео карте.

На диаграммах допускается ситуация когда один компонент зависит от двух или более других компонентов. Допускается взаимозависимость компонентов, когда оба компонента зависимы друг от друга.

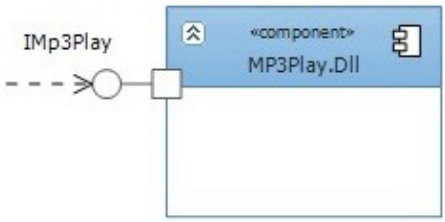
Линия зависимости не рисуется если независимый компонент является частью зависимого компонента.

Часто дизайнеры диаграмм подписывают зависимости, делается это для внесения большей ясности в диаграмму.

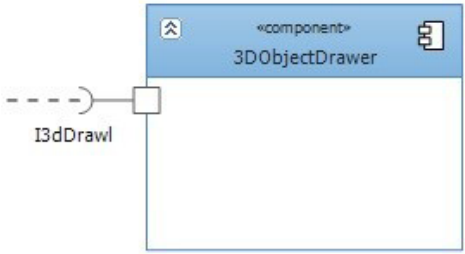
Интерфейсы – Важная часть диаграммы компонентов, часто бывает что компонент предоставляет для доступа к себе интерфейс.



Например: библиотека, которая проигрывает MP3 медиа файлы, предоставляет интерфейс, который показывает ее устройство (информацию о методах). Отображается данная ситуация следующим образом:



Второй случай, когда объект для своей работы требует объект поддерживающий интерфейс, например: компонент отображения на экране 3д объектов требует для своей работы компонент, который поддерживает интерфейс “I3dDraw” диаграмма будет нарисована следующим образом.



Чаще всего встречается, дует из этих двух способов описания интерфейса:

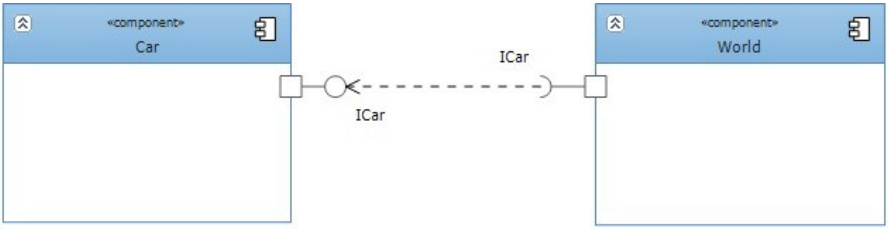
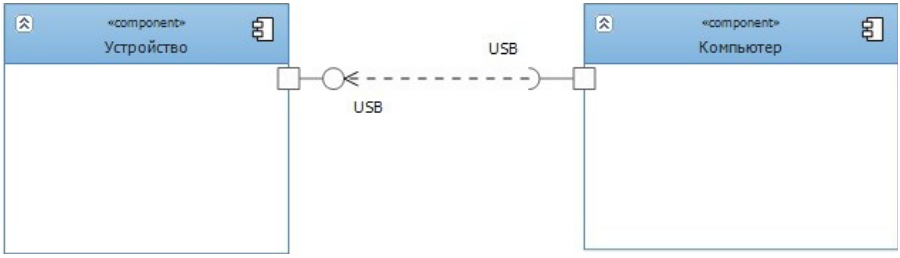


Диаграмма показывает, что компоненту мир для работы требуется компонент поддерживающий интерфейс ICar. Компонент Car поддердивает интерфейс ICar и это позволяет компоненту World использовать компонент Car

Более наглядным примером может служить компьютер и USB записывающие устройство:



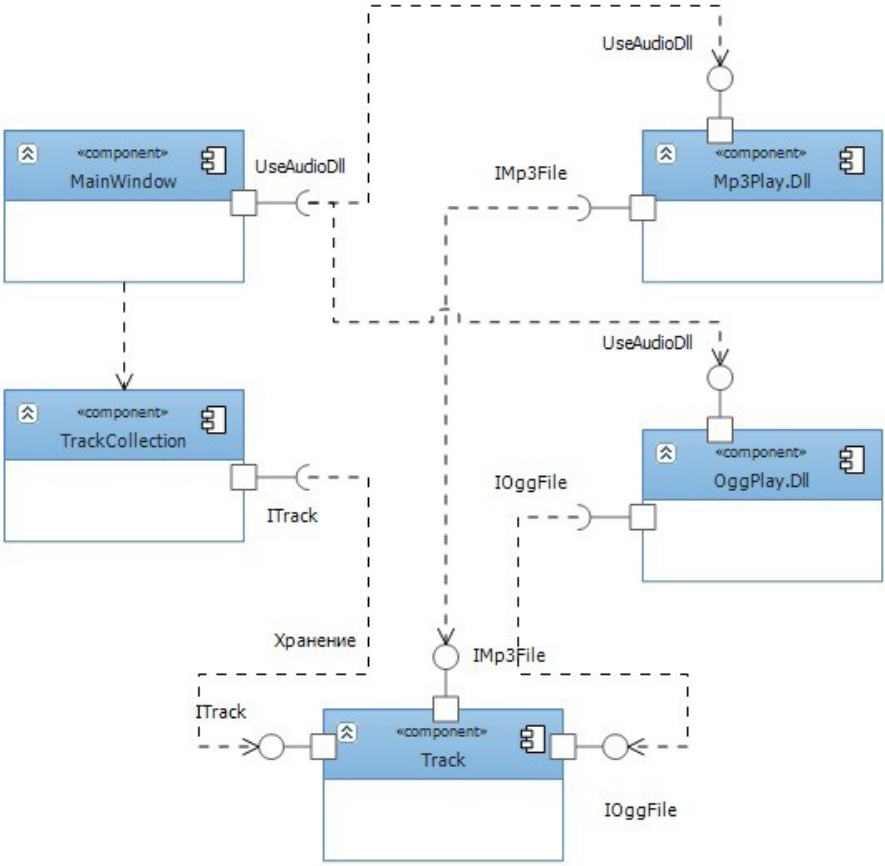
Обратите внимание на зависимости, когда объект для своей работы требует класс, поддерживающий определенный интерфейс это значит он зависим от класса, который поддерживает интерфейс.

3.3. Практические примеры построения

Для примера рассмотрим приложение: Медиа проигрыватель. Ознакомимся со всеми компонентами этого приложения:

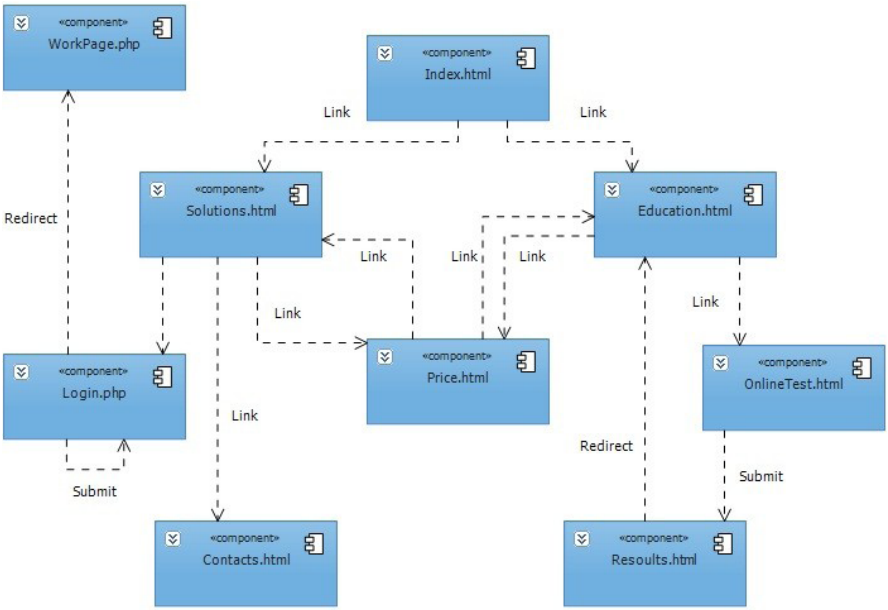
- MainWindow – пользовательский интерфейс программы, и вся ее основная функциональность, данный класс умеет пользоваться объектами унаследованными от интерфейса «UseAudioDll». Также данный компонент использует класс "TrackCollection" и умеет управлять им;
- TrackCollection – коллекция которая занимается хранением объектов поддерживающих интерфейс «ITrack», на диаграмме видно что данный объект нуждается в объектах поддерживающих данный интерфейс;
- Track – класс содержащий информацию о медиа треке, объекты данного класса поддерживают сразу три интерфейса «ITrack», «IMp3File», «IOggFile»;
- Mp3Play.Dll – библиотека для проигрывания файлов формата MP3;
- библиотека предоставляет доступ управления собой через интерфейс UseAudioDll но для корректной работы библиотеке требуется компонент поддерживающий интерфейс «IMp3File»;

- OggPlay.Dll – Библиотека имеющая аналогичные функции что и Mp3Play.Dll но для ее работы необходим компонент работающий посредством интерфейса IOggFile.



Возможно, вам покажется, что данную диаграмму лучше было бы изобразить в виде диаграммы классов, но это не так, диаграмма кооперации лучше показывает, каким образом происходит взаимодействие между компонентами приложения.

Далее мы рассмотрим диаграмму компонентов для корпоративного сайта



Когда строиться диаграмма компонентов для сайта стоит много внимания уделять именам зависимостей, с помощью их можно указать каким способом с одной страницы на другую будут передаваться данные, или каким способом пользователь будет переходить между ними.

4. Диаграмма развертывания

Диаграмма развертывания – используется для физического представления развертываемой системы. Проще говоря, диаграмма развертываний показывает, какие физические устройства и связи между ними необходимы для корректной работы разрабатываемого приложения.

Стоит отметить, что диаграмма развертывания иногда заменяется, на диаграмму компонентов.

4.1. Цели данного типа диаграмм

При составлении диаграммы развертывания можно найти слабые или не защищенные места в системе. Например: составив диаграмму развертывания для приложения онлайн игры, становиться ясно что сервер будет публичным, и должен быть хорошо защищен от хакерских атак, и иметь достаточное количество ресурсов для обслуживания большого количества клиентов.

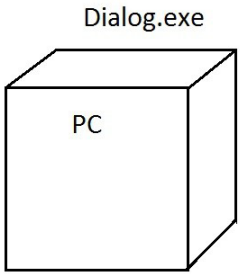
Построив диаграмму развертывания можно сказать, какие устройства будут необходимы для работы приложений, это позволит более конкретно понять суть разрабатываемого продукта.

4.2. Базовые понятия

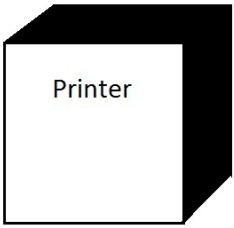
Диаграмма развертывания состоит из двух базовых понятий это «Узлы» и «Соединения».

Узел на диаграмме развертывания это какая либо вычислительная единица, как правило это устройство имеющее процессор, и (или) электронную, магнитную память. Узлом можно считать все, начиная от супер компьютера до USB запоминающего устройства.

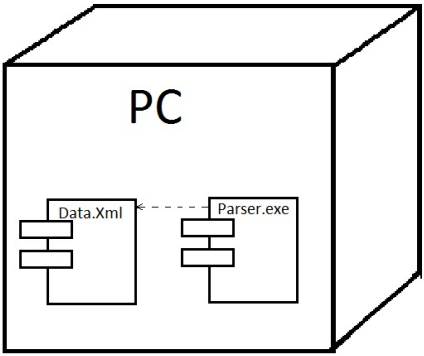
Узел изображается на диаграмме развертывания как куб.



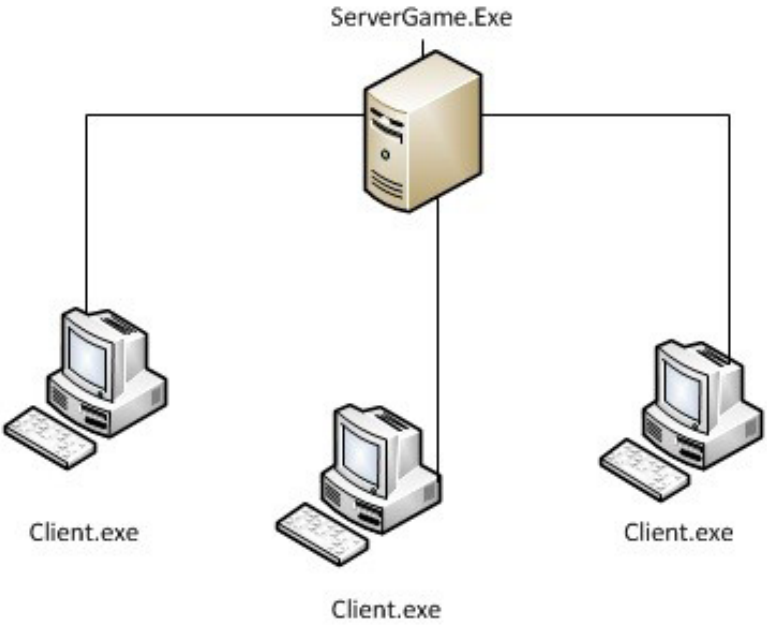
Таким образом, можно обозначить любой узел, необходимо написать на кубике тип данного устройства (возможно дополнительную информацию, например частоту процессора), сверху над кубиком обычно отображается имя компонента программы, которая работает на этом устройстве. Иногда для обозначения «простых» устройств (например, принтер) используется кубик с черными краями.



Также возможно соединить диаграмму компонентов с диаграммой развертывания, просто отобразив компоненты внутри устройства, например так:



Для построения диаграмм развертывания можно использовать Microsoft visio, на ней можно изображать узлы и соединения.

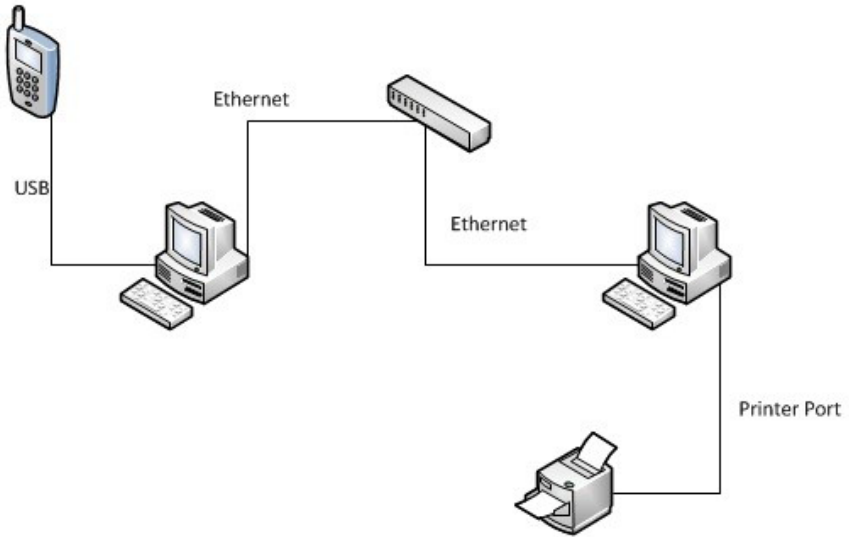


Узлом также является и человек, это позволяет графически изображать бизнес процессы.

Соединения на диаграмме развертывания изображаются простыми линиями без стрелок, как правило указывают на то что данный объект имеет отношение к объектом к которому ведет линия.

На показанной ниже диаграмме показано, что принтер связан только с одним компьютером, потому что линия ведет от принтера к одному из компьютеров.

Допускается подписание линий для более понятного представления диаграммы.

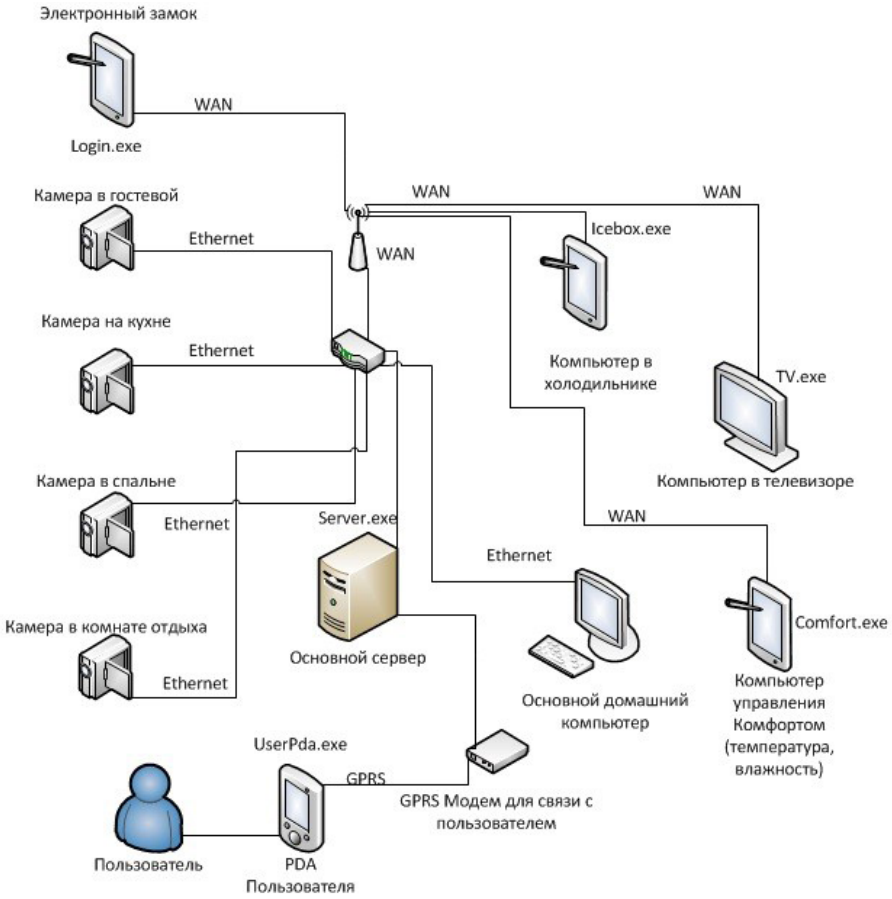


4.3. Практические примеры использования

Рассмотрим приложение управления системой «Умный дом».

Умный дом – жилое помещение оборудованное высокотехнологическими устройствами, которые помогают пользователю в хозяйстве. Хорошим примером такого устройства может служить холодильник который следит за количеством продуктов, и по мере надобности отправляет хозяину сообщение о том какие продукты испортились и им необходима замена.

Далее представлена диаграмма развертывания такой системы:



Дальше расписаны приложения и что они делают:

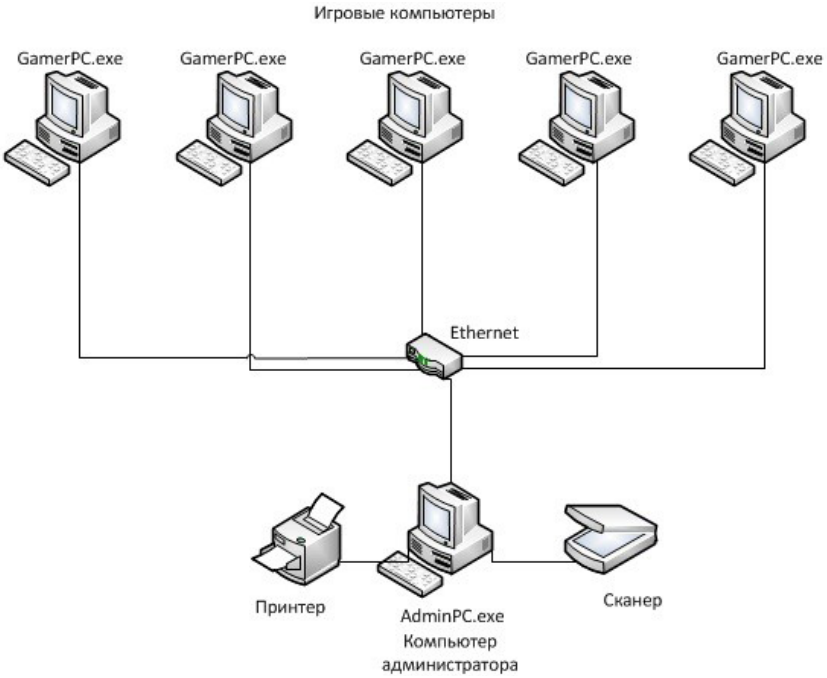
- Server.exe – Основное приложение установленное на узле «основной сервер» занимается сбором и хранением информации от каждого устройства, связан с GPRS модемом и роутером который транслирует
- Login.exe – Приложение электронный замок, просит пользователя ввести пароль, связывается с WiFi

точкой которая связывается с роутером который транслирует запрос на сервер, если сервер отправляет ответ что введенный пароль верный, электронный замок откроет дверь;

- Icebox.exe – приложение управляющее холодильником, связывается с сервером через WiFi который пускает сигнал транзитом через роутер к основному серверу;
- Tv.exe – приложение управляющее телевизором соединяется с сервером таким же образом как и компьютер в холодильнике;
- Comfort.exe – приложение для настройки климата в доме, соединяется с сервером так же как компьютер в холодильнике;
- UserPda.exe – приложение установлено в портативном компьютере у пользователя.

Проанализировав диаграмму можно сделать вывод, что роутер у входа на сервер довольно перегружен потому как будет передаваться потоковое видео из камер наблюдения. Возможно, стоит добавить компьютер, который будет занят хранением записей с камер. Также сеть имеет точку WiFi доступа, потому, необходимо создать защиту для приложений. Это не позволит, чтобы злоумышленник не смог поменять пользователю температуру в доме без ведома владельца.

Следующие приложение, представленное для рассмотрения, это программа для учета времени игры игроков в компьютерном клубе.



В данном продукте для компьютерных клубов будет находиться лишь два продукта:

- AdminPC.exe – приложение установленное н компьютере администратора, устанавливает количество времени до выключения пользовательских компьютеров;
- GamerPC.exe – приложение разрешает или не разрешает пользователю играть, решение принимается от того оплатил ли пользователь игру;

Проанализировав схему становиться, понято, что компьютер администратора будут больше всего подвержен атакам хакеров, так как именно с него начисляется время для пользователей. Компьютер администратора должен быть устойчив, простив вирусов.

5. Экзаменационное задание

В качестве экзаменационной работы Вам предлагается разработать модель информационной системы (приложения). К экзаменационной работе выдвигаются следующие требования:

1. Модель должна быть полной, то есть описывать всё множество компонентов и внутренних связей системы, а также вариантов использования этой системы пользователем и вариантов развёртывания;
2. Модель должна содержать всё множество необходимых диаграмм, описывающих все стороны функционирования моделируемой системы;
3. Работа должна содержать как минимум по одной диаграмме каждого типа;
4. При разработке модели необходимо руководствоваться принципами эффективности и целесообразности вносимых в модель компонентов и связей. Например, модульность необходимо предпочитать сложности. Другими словами, слишком сложные компоненты лучше разбивать на модули, с целью упрощения структуры каждого отдельного компонента и увеличения гибкости и масштабируемости общей структуры модули.

Далее следуют варианты экзаменационного задания:

1. Разработать модель системы учёта товаров супермаркета. Система должна поддерживать информационное обеспечение всех процессов, связанных с товарооборотом в супермаркете: учёт товаров на складе – информационное обеспечение процессов приёма, аудита и выдача товара на складе; размещение товаров в торговых помещениях – информационное обеспечение приёма товара на реализацию в торговое помещение со склада, учёт месторасположения товара в торговых помещениях и аудит работы сотрудников с товаром; программное обеспечение работы касс.
2. Разработать модель распределённого клиент-серверного приложения осуществляющего мониторинг рабочих станций и внутренней сетевой активности локальной сети, с последующим сбором сведений о состоянии каждой рабочей станции на сервере в виде совокупности лог-файлов. Приложение должно также поддерживать возможность настройки системы оповещения администратора о возникновении внештатных ситуаций, связанных с критическими нарушениями с точки зрения безопасности и целостности сети, мониторинг которой осуществляется (администратор должен иметь возможность настраивать систему оповещения, устанавливая список критериев безопасности и указывая критические значения для каждого из критериев).

top
КОМПЬЮТЕРНАЯ АКАДЕМИЯ

Урок №4

Диаграмма последовательности, диаграмма кооперации, диаграмма компонентов и диаграмма развертывания