Министерство образования Республики Беларусь
Министерство образования и науки Российской Федерации

Учреждение образования
БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ

Кафедра «Автоматизированные системы управления»

С.К. Крутолевич

«ПРОЕКТИРОВАНИЕ АСОИУ»

Конспект лекций

для студентов специальности
"Автоматизированные системы обработки информации"

Тема 1 Методология проектирования АСОИУ

Под системой обычно понимают множество элементов, находящихся в отношениях и связях друг с другом, которое образует определенную целостность, единство.

Первые представления о системе возникли в античной философий и пауке, выдвинувшей истолкование системы как упорядоченности и целостности бытия (Евклид, Платон, Аристотель и др.)

В современном научно-техническом знании разработка проблематики, связанной с исследованием и конструированием систем, проводится в рамках системного подхода, общей теории систем, различных специальных теорий систем, в кибернетике, системном анализе и т. д.

Представление о системе всегда связывается с такими понятиями, как элемент, целостность, структура, связь, отношение, подсистема. Исследователи разных областей знаний вкладывают в эти понятия; различный смысл. Общим, однако, для всех областей знании является то, что понятие «система» предполагает рассмотрение объекта как целого, состоящего из совокупности элементов. Любую систему можно расчленить (не обязательно единственным образом) на конечное число частей, называемых подсистемами, каждую из которых, в свою очередь, можно разделить на конечное число более мелких подсистем, вплоть до получения подсистем первого уровня — так называемых элементов системы. Элементом системы может быть не только какой-либо реальный объект или его часть, но и ряд его свойств.

Имеется несколько определений АСОИУ, незначительно отличающихся друг от друга:

1) система управления с применением современных автоматических средств обработки данных (ЭВМ, автоматических устройств накопления, регистрации, отображения и др.) и экономико-математических методов для решения основных задач управления производственно-хозяйственной деятельностью предприятия:

2) система управления производственно-хозяйственной деятельностью предприятия, органически включающая интегрированные системы обработки данных, главной целью которых является автоматизация процессов, сбора и переработки информации на предприятии и усовершенствование формы организации их выполнения;

3) система, в контуре управления которой функционируют совместно человек и технические средства, осуществляющие сбор и содержательную обработку информации по разветвленным алгоритмам с целью принятия оптимальных решений по управлению процессом или производством.

Но, несмотря на некоторые расхождения в определении АСОИ, бесспорно то, что автоматизированные системы управления — эго сложные человеко-машинные системы.

При создании таких систем разработчики сталкиваются с необходимостью рациональной организации и обеспечения взаимодействия большого числа разнородных составных частей. К ним относятся технические средства сбора, переработки, передачи и отображения информации, информационное, математическое и организационное обеспечение АСОИиУ. Все эти составные части АСОИиУ требуют привлечения специалистов различных областей знаний. Кроме того, работы в области автоматизации процессов управления требуют глубокого анализа сущности автоматизируемых процессов, выделения объектов автоматизации, правильного формулирования целей управления и определения этапности автоматизации с учетом реальных сроков, возможностей технической реализации и экономических факторов.

Объектом управления в АСОИиУ может быть рабочее место, отдел, участок, цех, предприятие, объединение и т. д. К объекту управления на уровне предприятия относят ту часть элементов, которые непосредственно участвуют в процессе материального производства и его обслуживания. К управляющей части системы относят множества элементов, необходимых для осуществления процесса управления объектом. Это управленческий персонал, техни-

ческие средства и методы управления.

Управляющая часть системы оперирует, как правило, с документами. Поэтому для эффективного управления производственной системой необходим непрерывный обмен информацией между объектом управления и управляющей частью.

Современные АСОИиУ представляют собой сложные человеко-машинные системы, в которых сочетается машинная переработка информации с координирующей деятельностью человека, принимающего решение.

При этом за человеком остаются, как правило, наиболее сложные, пока не поддающиеся формализации задачи, такие, например, как постановка проблемы, принятие решения в условиях неполной информации и неопределенности, контроль за переработкой информации.

Но главное—это система конечных и промежуточных целей, система критериев и ограничений, система, регламентирующая права, обязанности и меру ответственности, и, конечно, четко фиксированная система процедур принятия решения.

Функционирование АСОИиУ связано и с решением ряда проблем правового, а также психологического и социологического характера. Социологический аспект АСОИиУ имеет не меньшее значение, чем аспект структурный по отношению к отдельным уровням управления.

В основу разработки АСОИиУ должны быть положены следующие принципы: _новых задач, системного подхода, первого руководителя, непрерывного развития системы, автоматизации документооборота, согласованности пропускных способностей отдельных частей системы, типовости, однократности ввода данных и др.

Эффективность АСУ повышается при решении задач, которые при традиционной ручной технологии управления невозможно решить либо они решаются частично (принцип новых задач).

Второй принцип — это системный (комплексный) подход к созданию АСУ. Проектирование АСУ должно основываться на системном анализе как объекта, так и управляющей части. В частности, это означает, что следует определять цели и критерии для функционирования системы управления и проводить структурный анализ.

Принцип первого руководителя состоит в том, что разработку и внедрение АСУ нужно производить под непосредственным руководством первого руководителя соответствующего объекта. Отечественная и зарубежная практика свидетельствует, что всякая попытка передоверить дело создания АСУ второстепенным лицам неизбежно приводит к тому, что система ориентируется на рутинные задачи управления и не дает ожидаемого эффекта.

Согласно принципу непрерывного развития системы необходимо при проектировании АСУ предусмотреть возможность быстро реагировать на возникновение новых задач управления и совершенствование старых в процессе развития экономики и отдельных предприятий. .

Принцип автоматизации документооборота означает, что надо автоматизировать не только те или иные расчеты, но и оформление выходных документов, сбор исходных данных и в определенной мере передачу их и управляющих воздействий.

Согласно принципу согласованности пропускных способностей желательно равенство пропускных способностей последовательных звеньев. На практике добиваются соблюдения такого условия: пропускная способность каждого последующего звена должна быть не меньше пропускной способности предыдущего звена. Так, быстродействие ЭВМ следует увязать с требуемым количеством каналов связи, предназначенных для передачи информации.

Принцип типовости важен при разработке программ и сводится к максимальному использованию стандартных подпрограмм и типизации программ решения задач.

Важное значение для АСОИиУ имеет принцип однократности ввода данных в машину, согласно которому многократное использование любого рода сведений при решении задач на ЭВМ не должно приводить к повторному вводу каких то данных .

Особое значение для большинства АСОИиУ имеет проблема надежности. Для повышения надежности работы систем управления применяют высоконадежные технические средства, автоматическое резервирование важнейших узлов, и блоков. АСОИиУ должны об-

ладать также повышенной живучестью. Под живучестью АСУ понимают способность системы к определенной компенсации последствий нарушений и повреждений отдельных ее устройств, позволяющую системе продолжать выполнение основных функций при утрате пли временном снижении некоторых второстепенных показателей (точности, быстродействия, объема обрабатываемой информации) Для .повышения живучести системы целесообразно так распределять функции и соответственно потоки информации, чтобы на низшем уровне решалось иаибольшее количество задач местного характера, а на более высоких уровнях — задачи, имеющие общий характер Такое распределение функций упрощает проблемы замещения отдельных устройств другими такого же уровня или ЭВМ высшего уровня

Тема 2 Этапы проектирования АСОИУ.

Разделение процесса разработки сложных программных приложений на отдельные этапы способствовало становлению концепции жизненного цикла программы. Под жизненным циклом (ЖЦ) программы понимают совокупность взаимосвязанных и следующих во времени этапов, начиная от разработки требований к ней и заканчивая полным отказом от ее использования. Стандарт ISO/IEC 12207, хотя и описывает общую структуру ЖЦ программы, не конкретизирует детали выполнения тех или иных этапов. Согласно принятым взглядам ЖЦ программы состоит из следующих этапов:

- Анализа предметной области и формулировки требований к программе
- Проектирования структуры программы
- Реализации программы в кодах (собственно программирования)
- Внедрения программы
- Сопровождения программы
- Отказа от использования программы

На этапе анализа предметной области и .формулировки требований осуществляется определение функций, которые должна выполнять разрабатываемая программа, а также концептуализация предметной области. Эту работу выполняют аналитики совместно со специалистами предметной области. Результатом данного этапа должна являться некоторая концептуальная схема, содержащая описание основных компонентов и тех функций, которые они должны выполнять.

Этап проектирования структуры программы заключается в разработке детальной схемы будущей программы, на которой указываются классы, их свойства и методы, а также различные взаимосвязи между ними. Как правило, на этом этапе могут участвовать в работе аналитики, архитекторы и отдельные квалифицированные программисты. Результатом данного этапа должна стать детализированная схема программы, на которой указываются все классы и взаимосвязи между ними в процессе функционирования программы. Согласно методологии ООАП, именно данная схема должна "служить исходной информацией для написания программного кода.

Этап программирования вряд ли нуждается в уточнении, поскольку является наиболее традиционным для программистов. Появление инструментариев быстрой разработки приложений (Rapid Application Development, RAD) позволило существенно сократить время, и затраты на выполнение этого этапа. Результатом данного этапа является программное приложение, которое обладает требуемой функциональностью и способно решать нужные задачи в конкретной предметной области.

Этапы внедрения и сопровождения программы связаны с необходимостью настройки и конфигурирования среды программы, а также с устранением возникших в процессе ее использования ошибок. Иногда в качестве отдельного этапа выделяют тестирование программы, под которым понимают проверку работоспособности программы на некоторой совокупности исходных данных или при некоторых специальных режимах эксплуатации. Результатом этих этапов является повышение надежности Программного приложения, исключающе-

го возникновение критических ситуаций или нанесение ущерба компании, использующей данное приложение.

Рассматривая различные этапы ЖЦ программы, следует отметить одно важное обстоятельство. А именно, если появление RAD-инструментариев позволило существенно сократить сроки этапа программирования, то отсутствие соответствующих средств для первых двух этапов долгое время сдерживало процесс разработки приложений. Развитие методологии ООАП было направлено на автоматизацию второго, а затем и первого этапов ЖЦ программы.

Тема 3 Современные технологии проектирования АСОИУ.

Необходимость анализа предметной области до начала написания программы была осознана давно при разработке масштабных проектов. Процесс разработки баз данных существенно отличается от написания программного кода для решения вычислительной задачи. Главное отличие заключается в том, что при проектировании базы данных возникает необходимость в предварительной разработке концептуальной схемы, которая отражала бы общие взаимосвязи предметной области и особенности организации соответствующей информации. При этом под предметной областью принято понимать ту часть реального мира, которая имеет существенное значение или непосредственное отношение к процессу функционирования программы. Другими словами, предметная область включает в себя только те объекты и взаимосвязи между ними, которые необходимы для описания требований и условий решения некоторой задачи.

Выделение исходных или базовых компонентов предметной области, необходимых для решения той или иной задачи, представляет, в общем случае, нетривиальную проблему. Сложность данной проблемы проявляется в неформальном характере процедур или правил, которые можно применять для этой цели. Более того, такая работа должна выполняться совместно со специалистами или экспертами, хорошо знающими предметную область. Например, если разрабатывается база данных для обслуживания пассажиров крупного аэропорта, то в проектировании концептуальной схемы базы данных должны принимать участие штатные сотрудники данного аэропорта. Эти сотрудники должны хорошо знать весь процесс обслуживания пассажиров или данную предметную область.

Для выделения или идентификации компонентов предметной области было предложено несколько способов и правил. Сам этот процесс получил название концептуализации предметной области. При этом под компонентой понимают некоторую абстрактную единицу, которая обладает функциональностью, т. е. может выполнять определенные действия, связанные с решением поставленных задач. На предварительном этапе концептуализации рекомендуется использовать так называемые CRC-карточки (Component, Responsibility, Collaborator— компонента, обязанность, сотрудники). Для каждой выделенной компоненты предметной области разрабатывается собственная CRC-карточка (рис. 1.).

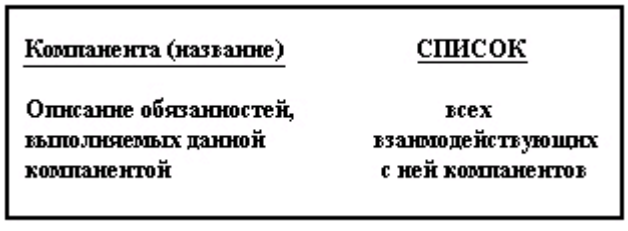

Рис. 1 Общий вид CRC-карточки для описания компонентов предметной области

Появление методологии ООАП потребовало, с одной стороны, разработки различных средств концептуализации предметной области, а с другой — соответствующих специалистов, которые владели бы этой методологией. На данном этапе появляется относительно но-

вый тип специалиста, который получил название аналитика или архитектора. Наряду со специалистами по предметной области аналитик участвует в построении концептуальной схемы будущей программы, которая затем преобразуется программистами в код. При этом отдельные компоненты выбираются таким образом, чтобы при последующей разработке их было удобно представить в форме классов и объектов. В этом случае немаловажное значение приобретает и сам язык представления информации о концептуальной схеме предметной области.

Методология ООАП тесно связана с концепцией автоматизированной разработки программного обеспечения (Computer Aided Software Engineering, CASE). Появление первых CASE-средств было встречено с определенной настороженностью. Со временем появились как восторженные Отзывы об их применении, так и критические оценки их возможностей. Причин для столь противоречивых мнений было несколько. Первая из них заключается в том, что ранние CASE-средства были простой надстройкой над некоторой системой управления базами данных (СУБД). Хотя визуализация процесса разработки концептуальной схемы БД имеет немаловажное значение, она не решает проблем разработки приложений других типов.

Вторая причина имеет более сложную природу, поскольку связана с графической нотацией, реализованной в том или ином CASE-средстве. Если языки программирования имеют строгий синтаксис, то попытки предложить подходящий синтаксис для визуального представления концептуальных схем БД были восприняты далеко неоднозначно. Появилось несколько подходов, которые более подробно будут рассмотрены познее На этом фоне появление унифицированного языка моделирования (Unified Modeling Language, UML), который ориентирован на решение задач первых двух этапов ЖЦ программ, было воспринято с большим оптимизмом всем сообществом корпоративных программистов.

Последнее, на что следует обратить внимание, это осознание необходимости построения предварительной модели программной системы, которую, согласно современным концепциям ООАП, следует считать результатом первых этапов ЖЦ программы. Поскольку язык UML даже в своем названии имеет отношение к моделированию, следует дополнительно остановиться на целом ряде достаточно важных вопросов. Таким образом, мы переходим к теме, которая традиционно не рассматривается в изданиях по ООАП, но имеющая самое прямое отношение к процессу построения моделей и, собственно, моделированию. Речь идет о методологии системного анализа и системного моделирования.

Системный анализ как научное направление имеет более давнюю историю, чем ООП и ООАП, и собственный предмет исследования. Центральным понятием системного анализа является понятие системы, под которой понимается совокупность объектов, компонентов или элементов произвольной природы, образующих некоторую целостность. Определяющей предпосылкой выделения некоторой совокупности как системы является возникновение у нее новых свойств, которых не имеют составляющие ее элементы. Примеров систем можно привести достаточно много — это персональный компьютер, автомобиль, человек, биосфера, программа и др. Более ортодоксальная точка зрения предполагает, что все окружающие нас предметы являются системами.

Важнейшими характеристиками любой системы являются ее структура и процесс функционирования. Под структурой системы понимают устойчивую во времени совокупность взаимосвязей между ее элементами или компонентами. Именно структура связывает воедино все элементы и препятствует распаду системы на отдельные компоненты. Структура системы может отражать самые различные взаимосвязи, в том числе и вложенность элементов одной системы в другую. В этом случае принято называть более мелкую или вложенную систему подсистемой, а более крупную — метасистемой.

Процесс функционирования системы тесно связан с изменением ее свойств или поведения во времени. При этом важной характеристикой системы является ее состояние, под

которым понимается совокупность свойств или признаков, которые в каждый момент времени отражают наиболее существенные особенности поведения системы.

Рассмотрим следующий пример. В качестве системы представим себе "Автомобиль". Для этого случая система охлаждения двигателя будет являться подсистемой "Автомобиля". С одной стороны, двигатель является элементом системы "Автомобиль". С другой стороны, двигатель сам является системой, состоящей из отдельных компонентов, таких как цилиндры, свечи зажигания и др. Поэтому система "Двигатель" также будет являться подсистемой системы "Автомобиль".

Структура системы "Автомобиль" может быть описана с разных точек зрения. Наиболее общее представление о структуре этой системы дает механическая схема устройства того или иного автомобиля. Взаимодействие элементов в этом случае носит механический характер. Состояние автомобиля можно рассматривать также с различных точек зрения, наиболее общей из которых является характеристика автомобиля как исправного или неисправного. Очевидно, что каждое из этих состояний в отдельных ситуациях может быть детализировано. Например, состояние "неисправный" может быть конкретизировано в состояния "неисправность двигателя", "неисправность аккумулятора", "отсутствие подачи топлива" и пр. Важно иметь четкое представление, что подобная детализация должна быть адекватна решаемой задаче.

Процесс функционирования системы отражает поведение системы во времени и может быть представлен как последовательное изменение ее состояний: Если система изменяет одно свое состояние на другое, то принято говорить, что система переходит из одного состояния в другое. Совокупность признаков или условий изменения состояний системы в этом случае называется переходом. Для системы с дискретными состояниями процесс функционирования может быть представлен в виде последовательности состояний с соответствующими переходами.

Методология системного анализа служит концептуальной основой системно-ориентированной декомпозиции предметной области. В этом случае исходными компонентами концептуализации являются системы и взаимосвязи между ними. При этом понятие системы является более общим, чем понятия классов и объектов в ООАП. Результатом системного анализа является построение некоторой модели системы или предметной области.

Понятие модели столь широко используется в повседневной жизни, что приобрело очень много смысловых оттенков. Это и "Дом моделей" известного кутюрье, и модель престижной марки автомобиля, и модель политического руководства, и математическая модель колебаний маятника. Применительно к программным системам нас будет интересовать только то понятие модели, которое используется в системном анализе. А именно, под моделью будем понимать некоторое представление о системе, отражающее наиболее существенные закономерности ее структуры и процесса функционирования и зафиксированное на некотором языке или в другой форме.

Примеров моделей можно привести достаточно много. Например, аэродинамическая модель гоночного автомобиля или проектируемого самолета, модель ракетного двигателя, модель колебательной .системы, модель системы электроснабжения региона, модель избирательной компании и др.

Общим свойством всех моделей является их подобие оригинальной системе или системе-оригиналу. Важность построения моделей заключается в возможности их использования для получения информации о свойствах или поведении системы-оригинала. При этом процесс построения и последующего применения моделей для получения информации о системе-оригинале получил название моделирование.

Термин "моделирование" имеет довольно много смысловых оттенков, например, моделирование одежды или моделирование прически. Не отрицая важности этих сфер творчества, следует отметить, "что все эти аспекты моделирования лежат за пределами книги. Рассмотрение особенностей языка UML связано с вопросами логического или информационного моделирования систем.

Наиболее общей моделью системы является так называемая модель "черного ящика". В этом случае система представляется в виде прямоугольника, внутреннее устройство которого скрыто от аналитика или неизвестно. Однако система не является полностью изолированной от внешней среды, поскольку последняя оказывает на систему некоторые информационные или материальные воздействия. Такие воздействия получили название входных воздействий. В свою очередь, система также оказывает на среду или другие системы определенные информационные или материальные воздействия, которые получили название выходных воздействий. Графически данная модель может быть изображена следующим образом (рис. 2).

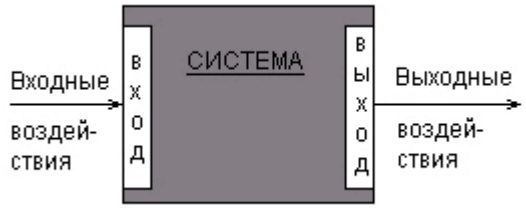

Рис. 2. Графическое изображение модели системы в виде "черного ящика"

Ценность моделей, подобных модели "черного ящика", весьма условна. Невольно может возникнуть ассоциация с "Черным квадратом". Однако если оценка изобразительных особенностей последнего не входит в задачи системного анализа, то общая модель системы содержит некоторую важную инфомацию о функциональных особенностях данной системы, которые дают представление о ее поведении. Действительно, кроме самой общей информации о том, на какие воздействия реагирует система, и как проявляется эта реакция на окружающие объекты и системы, другой информации мы получить не можем. В рамках системного анализа разработаны определенные методологические средства, позволяющие выполнить дальнейшую конкретизацию общей модели системы.

Процесс разработки адекватных моделей и их последующего конструктивного применения требует не только знания общей методологии системного анализа, но и наличия соответствующих изобразительных средств или языков для фиксации результатов моделирования и их документирования. Очевидно, что естественный язык не вполне подходит для этой цели, поскольку обладает неоднозначностью и неопределенностью. Для построения моделей были разработаны достаточно серьезные теоретические методы, основанные на развитии математических и логических средств моделирования, а также предложены различные формальные и графические нотации, отражающие специфику решаемых задач. Важно представлять, что унификация любого языка моделирования тесно связана с методологией системного моделирования, т. е. с системой воззрений и принципов рассмотрения сложных явлений и объектов как моделей сложных систем.

Сложность системы и, соответственно, ее модели может быть рассмотрена с различных точек зрения. Прежде всего, можно выделить сложность структуры системы, которая характеризуется количеством элементов системы и различными типами взаимосвязей между этими элементами. Если количество элементов превышает некоторое пороговое значение, которое не является строго фиксированным, то такая система может быть названа сложной. Например, если программная СУБД насчитывает более 100 отдельных форм ввода и вывода информации, то многие программисты сочтут ее сложной. Транспортная система современных мегаполисов также может служить примером сложной системы.

Вторым аспектом сложности является сложность процесса функционирования системы. Это может быть связано как с непредсказуемым характером поведения системы, так и невозможностью формального представления правил преобразования входных воздействий в выходные. В качестве примеров сложных программных систем можно привести современ-

ные операционные системы, которым присущи черты сложности как структуры, так и поведения.

Тема 4 Основные элементы языка UML

UML представляет собой общецелевой язык визуального моделирования, который разработан для спецификации, визуализации, проектирования и документирования компонентов программного обеспечения, бизнес- процессов и других систем. Он является простым и мощным средством моделирования, который может быть эффективно использован для построения концептуальных, логических и графических моделей сложных систем самого различного целевого назначения. Этот язык вобрал в себя наилучшие качества и опыт методов программной инженерии, которые с успехом использовались на протяжении последних лет при моделировании больших и сложных систем.

Язык UML основан на некотором числе базовых понятий, которые могут быть изучены и применены большинством программистов и разработчиков, знакомых с методами объектно-ориентированного анализа и проектирования. При этом базовые понятия могут комбинироваться и расширяться таким образом, что специалисты объектного моделирования получают возможность самостоятельно разрабатывать модели больших и сложных систем в самых различных областях приложений.

Конструктивное использование языка UML основывается на понимании общих принципов моделирования сложных систем и особенностей процесса объектно-ориентированного анализа и проектирования в частности. Выбор выразительных средств для построения моделей сложных систем предопределяет те задачи, которые могут быть решены с использованием данных моделей. При этом одним из основных принципов построения моделей сложных систем является принцип абстрагирования, который предписывает включать в модель только те аспекты проектируемой системы, которые имеют непосредственное отношение к выполнению системой своих функций или своего целевого предназначения. При этом все второстепенные детали опускаются, чтобы чрезмерно не усложнять процесс анализа и исследования полученной модели.

Другим принципом построения моделей сложных систем является принцип многомодельности. Этот принцип представляет собой утверждение о том, что никакая единственная модель не может с достаточной степенью адекватности описывать различные аспекты сложной системы. Феномен сложной системы как раз и состоит в том, что никакая ее единственная модель не является достаточной для адекватного выражения всех особенностей моделируемой системы.

Применительно к методологии ООАП это означает, что достаточно полная модель сложной системы представляет собой некоторое число взаимосвязанных представлений (views), каждое из которых адекватно отражает некоторый аспект поведения или структуры системы. При этом наиболее общими представлениями сложной системы принято считать статическое и динамическое представления, которые в свою очередь могут подразделяться на другие, более частные представления.

Еще одним принципом прикладного системного анализа является принцип иерархического построения моделей сложных систем. Этот принцип предписывает рассматривать процесс построения модели на разных уровнях абстрагирования или детализации в рамках фиксированных представлений. При этом исходная или первоначальная модель сложной системы имеет наиболее общее представление (метапредставление). Такая модель строится на начальном этапе проектирования и может не содержать многих деталей и аспектов моделируемой системы.

Рис. 3. Общая схема взаимосвязей моделей и представлений сложной системы в процессе объектно-ориентированного анализа и проектирования

Язык UML предназначен для решения следующих задач:

1. Предоставить в распоряжение пользователей легко воспринимаемый и выразительный язык визуального моделирования, специально предназначенный для разработки и документирования моделей сложных систем самого различного целевого назначения.

Речь идет о том, что важным фактором дальнейшего развития и повсеместного использования методологии ООАП является интуитивная ясность и понятность основных конструкций соответствующего языка моделирования. Язык UML включает в себя не только абстрактные конструкции для представления метамоделей систем, но и целый ряд конкретных понятий, имеющих вполне определенную семантику. Это позволяет языку UML одновременно достичь не только универсальности представления моделей для самых различных приложений, но и возможности описания достаточно тонких деталей реализации этих моделей применительно к конкретным системам.

Практика системного моделирования показала, что абстрактного описания языка на некотором метауровне недостаточно для разработчиков, которые ставят своей целью реализацию проекта системы в конкретные сроки. В настоящее время имеет место некоторый концептуальный разрыв между общей методологией моделирования сложных систем и конкретными инструментальными средствами быстрой разработки приложений. Именно этот разрыв по замыслу OMG и призван заполнить язык UML.

Отсюда вытекает важное следствие — для адекватного понимания базовых конструкций языка UML важно не только владеть некоторыми навыками объектно-ориентированного программирования, но и хорошо представлять себе общую проблематику процесса разработки моделей систем. Именно интеграция этих представлений образует новую парадигму ООАП, практическим следствием и центральным стержнем которой является язык UML.

2. Снабдить исходные понятия языка UML возможностью расширения и специализации для более точного представления моделей систем в конкретной предметной области.

Хотя язык UML является формальным языком — спецификаций, формальность его описания отличается от синтаксиса как традиционных формально-логических языков, так и известных языков программирования. Разработчики из OMG предполагают, что язык UML как никакой другой может быть приспособлен для конкретных предметных областей. Это становится возможным по той причине, что в самом описании языка UML заложен механизм расширения базовых понятий, который является самостоятельным элементом языка и имеет собственное описание в форме правил расширения.

В то же время разработчики из OMG считают крайне нежелательным переопределение базовых понятий языка по какой бы то ни было причине. Это может привести к неоднозначной интерпретации их семантики и возможной путанице. Базовые понятия языка UML

не следует изменять больше, чем это необходимо для их расширения. Все пользователи должны быть способны строить модели систем для большинства обычных приложений с использованием только базовых конструкций языка UML без применения механизма расширения. При этом новые понятия и нотации целесообразно применять только в тех ситуациях, когда имеющихся базовых понятий явно недостаточно для построения моделей системы.

Язык UML допускает также специализацию базовых понятий. Речь идет о том, что в конкретных7 приложениях пользователи должны уметь дополнять имеющиеся базовые понятия новыми характеристиками или свойствами, которые не противоречат семантике этих понятий в языке UML.

3. Описание языка UML должно поддерживать такую спецификацию моделей, которая не зависит от конкретных языков программирования и инструментальных средств проектирования программных систем.

Речь идет о том, что ни одна из конструкций языка UML не должна зависеть от особенностей ее реализации в известных языках программирования. То есть, хотя отдельные понятия языка UML и связаны с последними очень тесно, их жесткая интерпретация в форме каких бы то ни было конструкций программирования не может быть признана корректной. Другими словами, разработчики из OMG считают необходимым свойством языка UML его контекстно-программную независимость.

С другой стороны, язык UML должен обладать потенциальной возможностью реализации своих конструкций на том или ином языке программирования. Конечно, в первую очередь имеются в виду языки, поддерживающие концепцию ООП, такие как C++, Java, Object Pascal. Именно это свойство языка UML делает его современным средством решения задач моделирования сложных систем. В то же время, предполагается, что для программной поддержки конструкций языка UML могут быть разработаны специальные инструментальные CASE-средства. Наличие последних имеет принципиальное значение для широкого распространения и использования языка UML.

4. Описание языка UML должно включать в себя семантический базис для понимания общих особенностей ООАП.

Говоря об этой особенности, имеют в виду самодостаточность языка UML для понимания не только его базовых конструкций, но что не менее важно — понимания общих принципов ООАП. В этой связи необходимо отметить, что поскольку язык UML не является языком программирования, а служит средством для решения задач объектно-ориентированного моделирования систем, описание языка UML должно по возможности включать в себя все необходимые понятия для ООАП. Без этого свойства язык UML может оказаться бесполезным и невостребованным большинством пользователей, которые не знакомы с проблематикой ООАП сложных систем.

С другой стороны, какие бы то ни было ссылки на дополнительные источники, содержащие важную для понимания языка UML информацию, по мнению разработчиков из OMG, должны быть исключены. Это позволит избежать неоднозначного толкования принципиальных особенностей процесса ООАП и их реализации в форме базовых конструкций языка UML.

Пакеты в языке UML

Пакет — основной способ организации элементов модели в языке UML. Каждый пакет владеет всеми своими элементами, т. е. теми элементами, которые включены в него. Про соответствующие элементы пакета говорят, что они принадлежат пакету или входят в него. При этом каждый элемент может принадлежать только одному пакету. В свою очередь, одни пакеты могут быть вложены в другие пакеты. В этом случае первые называются подпакетами, поскольку все элементы подпакета будут принадлежать более общему пакету. Тем самым для элементов модели задается отношение вложенности пакетов, которое представляет собой иерархию.

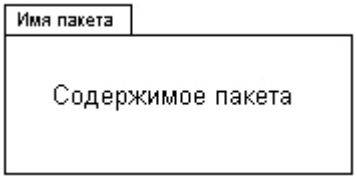

Рис. 4 Графическое изображение пакета в языке UML

В рамках языка UML все представления о модели сложной системы фиксируются в виде специальных графических конструкций, получивших название диаграмм. В терминах языка UML определены следующие виды диаграмм:

- Диаграмма вариантов использования (use case diagram)
- Диаграмма классов (class diagram)
- Диаграммы поведения (behavior diagrams)
  - Диаграмма состояний (statechart diagram)
  - Диаграмма деятельности (activity diagram)
  - Диаграммы взаимодействия (interaction diagrams)
    - Диаграмма последовательности (sequence diagram)
    - Диаграмма кооперации (collaboration diagram)
- Диаграммы реализации (implementation diagrams)
  - Диаграмма компонентов (component diagram)
  - Диаграмма развертывания (deployment diagram)

Интегрированная модель сложной системы в нотации UML показана на рисунке 5

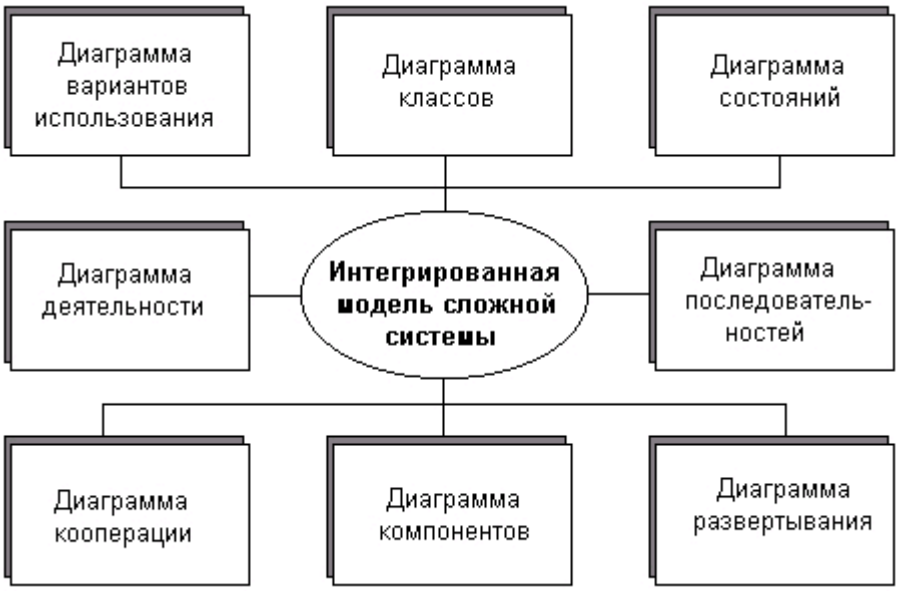

Рис. 5 Интегрированная модель сложной системы в нотации UML

Тема 5 Элементы графической нотации диаграммы вариантов использования.

Вариант использования представляет собой последовательность действий, выполняемых системой в ответ на событие, инициируемое некоторым внешним объектом (действующим лицом). Вариант использования описывает типичное взаимодействие между пользователем и системой. Вариант использования определяется в процессе обсуждения с пользователем тех функций, которые он хотел бы реализовать.

Действующее лицо (actor) – это роль, которую пользователь играет по отношению к системе. Действующие лица представляют собой роли, а не конкретных людей или наимено-

вания работ. Несмотря на то, что на диаграммах вариантов использования они изображаются в виде стилизованных человеческих фигурок, действующее лицо может также быть внешней системой, которой необходима некоторая информация от данной системы. Показывать на диаграмме действующих лиц следует только в том случае, когда им действительно необходимы некоторые варианты использования. Действующие лица делятся на три основных типа – пользователи системы; другие системы, взаимодействующие с данной; время. Время становится действующим лицом, если от него зависит запуск каких-либо событий в системе.

Все варианты использования так или иначе связаны с внешними требованиями к функциональности системы. Варианты использования всегда следует анализировать вместе с действующими лицами системы, определяя при этом реальные задачи пользователей и рассматривая альтернативные способы решения этих задач.

Конкретная цель диаграмм вариантов использования – это документирование вариантов использования (всё, входящее в сферу применения системы), действующих лиц (всё вне этой сферы) и связей между ними.

Разрабатывая диаграммы вариантов использования, старайтесь придерживаться следующих правил.

Не моделируйте связи между действующими лицами. По определению действующие лица находятся вне сферы действия системы. Это означает, что связи между ними также не относятся к её компетенции.

Не соединяйте сплошной стрелкой (коммуникационной связью) два варианта использования непосредственно. Диаграммы данного типа описывают, какие варианты использования доступны системе, а не порядок их выполнения. Для отображения порядка выполнения вариантов использования применяют диаграммы деятельности.

Вариант использования должен быть инициирован действующим лицом. Это означает, что должна быть сплошная стрелка, начинающаяся на действующем лице и заканчивающаяся на варианте использования.

Для детального описания диаграммы используется документ, называемый «поток событий» (flow of events). Целью потока событий является документирование процесса обработки данных, реализуемого в рамках варианта использования. Этот документ подробно описывает, что будут делать пользователи системы и что – сама система.

Данное описание не должен зависеть от реализации. Цель – описать, что будет делать система, а не как она будет это делать. Обычно поток событий включает:

– краткое описание;

– предусловия (pre-conditions);

– основной поток событий;

– альтернативный поток событий (или несколько альтернативных потоков);

– постусловия (post-conditions).

Последовательно рассмотрим эти составные части.

Описание.

Каждый вариант использования должен иметь связанное с ним короткое описание того, что он будет делать.

Предусловия.

Предусловия варианта использования – это такие условия, которые должны быть выполнены, прежде чем вариант использования начнет выполняться сам. Например, таким условием может быть выполнение другого варианта использования или наличие у пользователя прав доступа, требуемых для запуска этого. Не у всех вариантов использования бывают предварительные условия.

Ранее упоминалось, что диаграммы вариантов использования не должны отражать порядок их выполнения. С помощью предусловий, однако, можно документировать и такую информацию. Например, предусловием одного варианта использования может быть то, что в это время должен выполняться другой.

Основной и альтернативный потоки событий.

Потоки событий поэтапно описывают, что должно происходить во время выполнения заложенной в варианты использования функциональности. Поток событий уделяет внимание тому, что будет делать система, а не как она будет делать это, причем описывает все это с точки зрения пользователя. Основной и альтернативный потоки событий включают следующее описание:

– способ запуска варианта использования;

– различные пути выполнения варианта использования;

– нормальный, или основной, поток событий варианта использования;

– отклонения от основного потока событий (так называемые альтернативные потоки);

– потоки ошибок;

– способ завершения варианта использования.

Постусловия.

Постусловиями называются такие условия, которые всегда должны быть выполнены после завершения варианта использования. Например, в конце варианта использования можно пометить флажком какой-нибудь переключатель. Информация такого типа входит в состав постусловий. Как и для предусловий, с помощью постусловий можно вводить информацию о порядке выполнения вариантов использования системы. Если, например, после одного из вариантов использования должен всегда выполняться другой, это можно описать как постусловие. Такие условия имеются не у каждого варианта использования.

Связи между вариантами использования и действующими лицами.

На диаграммах вариантов использования поддерживается несколько типов связей между элементами диаграммы. Это связи коммуникации (communication), включения (include), расширения (extend) и обобщения (generalization).

Связь коммуникации – это связь между вариантом использования и действующим лицом. Связи коммуникации показывают с помощью однонаправленной ассоциации (сплошной линии).

Связь включения применяется в тех ситуациях, когда имеется какой-либо фрагмент поведения системы, который повторяется более чем в одном варианте использования. С помощью таких связей обычно моделируют многократно используемую функциональность.

Связь расширения применяется при описании изменений в нормальном поведении системы. Она позволяет варианту использования только при необходимости использовать функциональные возможности другого.

Связи включения и расширения показывают в виде штрихпунктирных линий со стрелками с соответствующими стереотипами.

С помощью связи обобщения показывают, что у нескольких действующих лиц имеются общие черты.

Варианты использования являются необходимым средством на стадии формирования требований к АСОИ. Каждый вариант использования – это потенциальное требование к системе, и пока оно не выявлено, невозможно запланировать ее реализацию.

Пример диаграммы показан на рис. 6

Рис. 6 Диаграмма вариантов использования для примера системы продажи товаров по каталогу

Тема 6 Спецификация требований и рекомендации по написанию эффективных вариантов использования.

Главное назначение диаграммы вариантов использования заключается в формализации функциональных требований к системе с помощью понятий соответствующего пакета и возможности согласования полученной модели с заказчиком на ранней стадии проектирования. Любой из вариантов использования может быть подвергнут дальнейшей декомпозиции на множество подвариантов использования отдельных элементов, которые образуют исходную сущность. Рекомендуемое общее количество актеров в модели — не более 20, а вариантов использования — не более 50. В противном случае модель теряет свою наглядность и, возможно, заменяет собой одну из некоторых других диаграмм.

Семантика построения диаграммы вариантов использования должна определяться следующими особенностями рассмотренных выше элементов модели. Отдельный экземпляр варианта использования по своему содержанию является выполнением последовательности действий, которая инициализируется посредством экземпляра сообщения от экземпляра актера. В качестве отклика или ответной реакции на сообщение актера экземпляр варианта использования выполняет последовательность действий, установленную для данного варианта использования. Экземпляры актеров могут генерировать новые экземпляры сообщений для экземпляров вариантов использования.

Подобное взаимодействие будет продолжаться до тех пор, пока не закончится выполнение требуемой последовательности действий экземпляром варианта использования, и соответствующий экземпляр актера (и никакой другой) не получит требуемый экземпляр сервиса. Окончание взаимодействия означает отсутствие инициализации экземпляров сообщений от экземпляров актеров для соответствующих экземпляров вариантов использования.

Варианты использования могут быть специфицированы в виде текста, а в последующем — с помощью операций и методов вместе с атрибутами, в виде графа деятельности, посредством автомата или любого другого механизма описания поведения, включающего предусловия и постусловия. Взаимодействие между вариантами использования и актерами мо-

жет уточняться на диаграмме кооперации, когда описываются взаимосвязи между сущностью, содержащей эти варианты использования, и окружением или внешней средой этой сущности.

В случае, когда для представления иерархической структуры проектируемой системы используются подсистемы, система может быть определена в виде вариантов использования на всех уровнях. Отдельные подсистемы или классы могут выступать в роли таких вариантов использования. При этом вариант, соответствующий некоторому из этих элементов, в последующем может уточняться множеством более мелких вариантов использования, каждый из которых будет определять сервис элемента модели, содержащийся в сервисе исходной системы. Вариант использования в целом может рассматриваться как суперсервис для уточняющих его подвариантов, которые, в свою очередь, могут рассматриваться как подсервисы исходного варианта использования.

Функциональность, определенная для более общего варианта использования, полностью наследуется всеми вариантами нижних уровней. Однако следует заметить, что структура элемента-контейнера не может быть представлена вариантами использования, поскольку они могут представлять только функциональность отдельных элементов модели. Подчиненные варианты использования кооперируются для совместного выполнения суперсервиса варианта использования верхнего уровня. Эта кооперация также может быть представлена на диаграмме кооперации в виде совместных действий отдельных элементов модели.

Отдельные варианты использования нижнего уровня могут участвовать в нескольких кооперациях, т. е. играть определенную роль при выполнении сервисов нескольких вариантов верхнего уровня. Для отдельных таких кооперация могут быть определены соответствующие роли актеров, взаимодействующих с конкретными вариантами использования нижнего уровня. Эти роли будут играть актеры нижнего уровня модели системы. Хотя некоторые из таких актеров могут быть актерами верхнего уровня, это не противоречит принятым в языке UML семантическим правилам построения диаграмм вариантов использования. Более того, интерфейсы вариантов использования верхнего уровня могут полностью совпадать по своей структуре с соответствующими интерфейсами вариантов нижнего уровня.

Окружение вариантов использования нижнего уровня является самостоятельным элементом модели, который в свою очередь содержит другие элементы модели, определенные для этих вариантов использования. Таким образом, с точки зрения общего представления верхнего уровня взаимодействие между вариантами использования нижнего уровня определяет результат выполнения сервиса варианта верхнего уровня. Отсюда следует, что в языке UML вариант использования является элементом-контейнером.

Варианты использования классов соответствуют операциям этого класса, поскольку сервис класса является по существу выполнением операций данного класса. Некоторые варианты использования могут соответствовать применению только одной операции, в то время как другие — конечного множества операций, определенных в виде последовательности операций. В то же время одна операция может быть необходима для выполнения нескольких сервисов класса и поэтому будет появляться в нескольких вариантах использования этого класса.

Реализация варианта использования зависит от типа элемента модели, в котором он определен. Например, поскольку варианты использования класса определяются посредством операций этого класса, они реализуются соответствующими методами. С другой стороны, варианты использования подсистемы реализуются элементами, из которых состоит данная подсистема. Поскольку подсистема не имеет своего собственного поведения, все предлагаемые подсистемой сервисы должны представлять собой композицию сервисов, предлагаемых отдельными элементами этой подсистемы, т. е., в конечном итоге, классами. Эти элементы могут взаимодействовать друг с другом для совместного обеспечения требуемого поведения отдельного варианта использования. Такое совместное обеспечение требуемого поведения описывается специальным элементом языка UML — кооперация или сотрудничество, который будет рассмотрен в главе 9, посвященной построению диаграмм кооперации. Здесь лишь

отметим, что кооперации используются как для уточнения спецификаций в виде вариантов использования нижних уровней диаграммы, так и для описания особенностей их последующей реализации.

Если в качестве моделируемой сущности выступает система или подсистема самого верхнего уровня, то отдельные пользователи вариантов использования этой системы моделируются актерами. Такие актеры, являясь внутренними по отношению к моделируемым подсистемам нижних уровней, часто в явном виде не указываются, хотя и присутствуют неявно в модели подсистемы. Вместо этого варианты использования непосредственно обращаются к тем модельным элементам, которые содержат в себе подобные неявные актеры, т. е. экземпляры которых играют роли таких актеров при взаимодействии с вариантами использования. Эти модельные элементы могут содержаться в других пакетах или подсистемах. В последнем случае роли определяются в том пакете, к которому относится соответствующая подсистема.

С системно-аналитической точки зрения построение диаграммы вариантов использования специфицирует не только функциональные требования к проектируемой системе, но и выполняет исходную структуризацию предметной области. Последняя задача сочетает в себе не только следование техническим рекомендациям, но и является в некотором роде искусством, умением выделять главное в модели системы. Хотя рациональный унифицированный процесс не исключает итеративный возврат в последующем к диаграмме вариантов использования для ее модификации, не вызывает сомнений тот факт, что любая подобная модификация потребует, как по цепочке, изменений во всех других представлениях системы. Поэтому всегда необходимо стремиться к возможно более точному представлению модели именно в форме диаграммы вариантов использования.

Если же варианты использования применяются для спецификации части системы, то они будут эквивалентны соответствующим вариантам использования в модели подсистемы для части соответствующего пакета. Важно понимать, что все сервисы системы должны быть явно определены на диаграмме вариантов использования, и никаких других сервисов, которые отсутствуют на данной диаграмме, проектируемая система не может выполнять по определению. Более того, если для моделирования реализации системы используются сразу несколько моделей (например, модель анализа и модель проектирования), то множество вариантов использования всех пакетов системы должно быть эквивалентно множеству вариантов использования модели в целом.

Тема 7 Элементы графической нотации диаграммы классов.

Центральное место в методологии ООАП занимает разработка логической модели системы в виде диаграммы классов.

Диаграмма классов (class diagram) служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов состоит из множества элементов, которые в совокупности отражают декларативные знания о предметной области. Эти знания интерпретируются в базовых понятиях языка UML, таких, как классы, интерфейсы и отношения между ними и составляющими их элементами. Отдельные элементы этой диаграммы могут организовываться в пакеты для представления более общей модели системы.

В общем случае пакет статической структурной модели может быть представлен в виде одной или нескольких диаграмм классов.

Декомпозиция подобного представления на отдельные диаграммы выполняется с целью удобства и графической визуализации структурных взаимосвязей предметной области.

Класс (class) является абстрактным описанием или представлением свойств множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов. Графически класс изображается в виде прямоугольника, который

разделен горизонтальными линиями на секции. В этих секциях указывается имя класса, атрибуты (переменные) и операции (методы).

Обязательным элементом обозначения класса является его имя. На начальных этапах разработки диаграммы указывают только имя класса. По мере проработки модели АСОИ диаграммы описания классов дополняются атрибутами и операциями. Окончательный вариант диаграммы содержит наиболее полное описание классов, которые состоят из трех секций.

Имя класса должно быть уникальным в пределах пакета, который может содержать несколько диаграмм классов. Имя класса записывается по центру самой верхней секции полужирным шрифтом и должно начинаться с заглавной буквы. В качестве имен классов используются существительные, записанные без пробелов. Необходимо помнить, что именно имена классов образуют словарь предметной области при ООАП. В секции имени класса могут также находиться стереотипы или ссылки на стандартные шаблоны, от которых образован данный класс и, соответственно, от которых он наследует атрибуты и операции.

Иногда необходимо явно указать, к какому пакету относится тот или иной класс. Для этой цели используется специальный символ-разделитель (двойное двоеточие ::). Синтаксис строки имени класса в этом случае будет следующий: <имя пакета>:: <имя класса>.

Атрибут (attribute) класса служит для представления отдельного свойства или признака, который является общим для всех объектов данного класса. Атрибуты класса записываются во второй сверху секции прямоугольника класса, поэтому эту секцию часто называют секцией атрибутов. В языке UML принята определенная стандартизация записи атрибутов класса. Каждому атрибуту класса соответствует отдельная строка текста, которая состоит из квантора видимости, имени, его кратности, типа значений атрибута и, возможно, его исходного значения.

Общий формат записи отдельного атрибута класса следующий:

<квантор видимости> <имя атрибута> [кратность]:<тип атрибута> = <исходное значение> {строка-свойство}

Квантор видимости (visibility) может принимать одно из четырех возможных значений и, соответственно, отображается при помощи специальных символов.

Символ «+» обозначает атрибут с областью видимости типа общедоступный (public). Атрибут с этой областью видимости доступен или виден из любого другого класса пакета, в котором определена диаграмма.

Символ «#» обозначает атрибут с областью видимости типа защищенный (protected). Атрибут с этой областью видимости недоступен или невиден для всех классов, за исключением подклассов данного класса.

Символ «—» обозначает атрибут с областью видимости типа закрытый (private). Атрибут с этой областью видимости недоступен или невиден для всех классов без исключения.

И, наконец, символ «~» обозначает атрибут с областью видимости типа пакетный (package). Атрибут с этой областью видимости недоступен или невиден для всех классов за пределами пакета, в котором определен класс-владелец данного атрибута.

Вместо условных графических обозначений можно записывать соответствующее ключевое слово: public, protected, private или package.

Имя атрибута представляет собой строку текста, которая используется в качестве идентификатора соответствующего атрибута и поэтому должна быть уникальной в пределах данного класса. Имя атрибута является единственным обязательным элементом синтаксического обозначения атрибута, оно должно начинаться со срочной (малой) буквы и не должно содержать пробелов.

Кратность атрибута характеризует общее количество конкретных атрибутов данного типа, входящих в состав отдельного класса. В общем случае кратность записывается в форме строки из цифр в квадратных скобках после имени соответствующего атрибута, при этом цифры разделяются двоеточием: [нижняя граница верхняя граница], где нижняя граница и верхняя граница являются положительными целыми числами. В качестве верхней границы

может использоваться специальный символ «*» (звездочка), который означает произвольное положительное целое число.

Тип атрибута определяется типом данных, определенным в пакете «Типы данных» языка UML или разработчиком. В нотации UML тип атрибута определяется в зависимости от языка программирования, который предполагается использовать для реализации данной модели. Типу атрибута должно предшествовать двоеточие.

Исходное значение служит для задания некоторого начального значения для соответствующего атрибута в момент создания отдельного экземпляра класса.

При задании атрибутов могут быть использованы две дополнительные синтаксические конструкции — это подчеркивание строки атрибута и пояснительный текст в фигурных скобках.

Подчеркивание строки атрибута означает, что соответствующий атрибут является общим для всех объектов данного класса.

Строка–свойство служит для указания дополнительных свойств атрибута, которые могут характеризовать особенности изменения значений атрибута в ходе выполнения соответствующей программы. Фигурные скобки обозначают фиксированное значение соответствующего атрибута для класса в целом, которое должны принимать все вновь создаваемые экземпляры класса без исключения.

Можно привести следующие примеры задания имен и типов атрибутов классов:

–имяСотрудника[1..2]:String=”Иван Иванович” —здесь имяСотрудника является именем атрибута, который служит для представления информации об имени сотрудника. Тип атрибута string как раз и указывает на тот факт, что отдельное значение имени представляет собой строку текста из одного или двух слов. По умолчанию значение данного поля “Иван Иванович”;

–видимость:Boolean= истина{frozen} — в данном случае видимость является именем атрибута. Тип Boolean означает, что возможными значениями данного атрибута являются одно из двух логических значений: истина (true) или ложь (false). В момент создания экземпляра класса данное свойство имеет значение «истина». Данное свойство не подлежит изменению с течением времени.

Операция (operation) — это некоторый сервис, который предоставляет каждый экземпляр или объект класса по требованию своих клиентов. Операции класса записываются в третьей сверху секции прямоугольника класса, поэтому эту секцию часто называют секцией операций. Совокупность операций характеризует функциональный аспект поведения всех объектов данного класса. Каждой операции класса соответствует отдельная строка, которая состоит из квантора видимости операции, имени, выражения типа возвращаемого значения.

Общий формат записи отдельной операции класса следующий:

<квантор видимости> <имя операции>(список параметров): <выражение типа возвращаемого значения>

Имя операции представляет собой строку текста, которая используется в качестве идентификатора соответствующей операции и поэтому должна быть уникальной в пределах данного класса. Имя операции является единственным обязательным элементом синтаксического обозначения операции, должно начинаться со срочной (малой) буквы и не должно содержать пробелов.

Список параметров является перечнем разделенных запятой формальных параметров, каждый из которых, в свою очередь, может быть представлен в следующем виде:

<вид параметра> <имя параметра> : <выражение типа> = <значение параметра по умолчанию>

Здесь вид параметра — одно из ключевых слов in, out или inout со значением in по умолчанию, в случае, если он не указывается. Имя параметра – это идентификатор соответствующего формального параметра, при записи которого следуют правилам задания имен атрибутов. Выражение типа – это спецификация типа данных для возвращаемого значения соответствующего формального параметра. Значение по умолчанию – это некоторое кон-

кретное значение для этого формального параметра.

Выражение типа возвращаемого значения также указывает на тип данных значения, которое возвращается объектом после выполнения соответствующей операции. Двоеточие и выражение типа возвращаемого значения могут быть опущены, если операция не возвращает никакого значения. Для указания нескольких возвращаемых значений данный элемент спецификации операции может быть записан в виде списка отдельных выражений.

Обязательной частью строки записи операции является наличие имени операции и круглых скобок. Список формальных параметров и тип возвращаемого значения могут не указываться. В качестве примеров записи операций можно привести несколько обозначений отдельных операций:

– «—»изменитьСчетКлиента (номерСчета : Integer) : Currency обозначает закрытую операцию с аргументом «номерСчета», который записывается в виде целого числа. Результатом выполнения этой операции является некоторое число, записанное в принятом денежном формате;

– «#» выдатьСообщение ( ):{“Ошибка деления на ноль”}. Данное сообщение может появиться на экране монитора в случае попытки деления некоторого числа на ноль.

Кроме внутреннего устройства или структуры классов важную роль при разработке проектируемой системы имеют различные отношения между классами, которые также могут быть изображены на диаграмме классов. Однако совокупность допустимых типов таких отношений строго фиксирована. Базовыми отношениями на диаграммах классов являются:

– отношение ассоциации (association relationship);
– отношение обобщения (generalization relationship);
– отношение агрегации (aggregation relationship);
– отношение композиции (composition relationship);
– отношение зависимости (dependency relationship).

Отношение ассоциации соответствует наличию произвольного отношения между классами. Данное отношение обозначается сплошной линией со стрелкой или без нее. В качестве дополнительных специальных символов могут использоваться имя ассоциации, символ навигации, а также имена и кратность классов-ролей ассоциации.

Имя ассоциации является необязательным элементом ее обозначения. Однако если оно задано, то записывается с заглавной буквы рядом с линией соответствующей ассоциации. Отдельные классы ассоциации могут играть некоторую роль в соответствующем отношении, что может быть явно указано на диаграмме заданием имени концевых точек ассоциации.

Наиболее простой случай данного отношения — бинарная ассоциация. Она связывает в точности два различных класса и может быть ненаправленным (симметричным) или направленным отношением.

Ненаправленная бинарная ассоциация изображается линией без стрелки. Для нее на диаграмме может быть указан порядок чтения классов с использованием значка в форме треугольника рядом с именем данной ассоциации.

Направленная бинарная ассоциация изображается сплошной линией с простой стрелкой на одном из ее концевых точек. Направление этой стрелки указывает на то, какой из классов является первым, а какой — вторым (на него указывает стрелка ассоциации).

Следующий элемент обозначений — кратность ассоциации. Кратность относится к концам ассоциации и обозначается в виде интервала целых чисел, аналогично кратности атрибутов и операций классов, но без прямых скобок. Этот интервал записывается рядом с концом соответствующей ассоциации и означает потенциальное число отдельных экземпляров класса, которые могут иметь место.

Ассоциация является наиболее общей формой отношения в языке UML. Все другие типы рассматриваемых отношений можно считать частным случаем данного отношения.

Отношение обобщения является отношением между более общим элементом (родителем) и более частным элементом (дочерним). Данное отношение может использоваться для

представления иерархических взаимосвязей между пакетами, классами, вариантами использования и другими элементами языка UML.

Применительно к диаграмме классов данное отношение описывает наследование их свойств и поведения. Согласно одному из главных принципов методологии ООАП — наследованию, класс-потомок обладает всеми свойствами и поведением класса-предка, а также имеет свои собственные свойства и поведение, которые могут отсутствовать у класса-предка. На диаграммах отношение обобщения обозначается сплошной линией с треугольной стрелкой на одном из концов. Стрелка указывает на более общий класс (родитель).

В дополнение к простой стрелке–обобщению может быть присоединена строка текста, указывающая на некоторые специальные свойства этого отношения. Этот текст будет относиться ко всем линиям обобщения, которые идут к классам-потомкам.

В качестве ограничений могут быть использованы следующие ключевые слова языка UML:

– (complete) означает, что в данном отношении обобщений специфицированы все классы-потомки и других классов-потомков у данного класса-предка быть не может;

– (incomplete) означает случай, противоположный первому, т. е. предполагается, что на диаграмме указаны не все классы-потомки. В последующем, возможно, разработчик восполнит их перечень, не изменяя уже построенную диаграмму.

Отношение агрегации имеет место между несколькими классами в том случае, если один из классов включает в себя в качестве составных частей другие классы.

Данное отношение применяется для представления системных взаимосвязей типа «часть–целое». Раскрывая внутреннюю структуру системы, отношение агрегации показывает, из каких элементов состоит система и как они связаны между собой. Графически отношение агрегации изображается сплошной линией, один из концов которой представляет собой не закрашенный внутри ромб. Этот ромб указывает на тот из классов, который представляет собой класс-контейнер. Остальные классы являются его «частями».

Отношение композиции является частным случаем отношения агрегации. Специфика этой взаимосвязи заключается в том, что части не могут выступать в отрыве от целого, т. е. с уничтожением целого уничтожаются и все его составные части. Графически отношение композиции изображается сплошной линией, один из концов которой представляет собой закрашенный внутри ромб. Этот ромб указывает на тот из классов, который представляет собой класс–композит. Остальные классы являются его «частями».

Отношение зависимости используется в такой ситуации, когда некоторое изменение одного элемента модели может потребовать изменения другого, зависящего от него элемента модели. Отношение зависимости графически изображается пунктирной линией между соответствующими элементами со стрелкой на одном из ее концов. На диаграмме классов данное отношение связывает отдельные классы между собой, при этом стрелка направлена от класса-клиента зависимости или зависимого класса к классу-источнику или независимому классу.

Наиболее часто в качестве классов-клиентов и классов-источников зависимости могут выступать целые множества классов, организованные в пакеты. В этом случае стрелка должна быть направлена от пакета-клиента зависимости или зависимого пакета к пакету-источнику или независимому пакету.

При моделировании предметной области различают следующие виды классов.

Управляющий класс (control class), отвечающий за координацию действий других классов. У каждого варианта использования и диаграммы классов должен быть хотя бы один управляющий класс, контролирующий последовательность выполнения действий этого варианта использования. Этот класс является активным и инициирует рассылку множества сообщений другим классам модели. Кроме специального обозначения управляющий класс может быть изображен в форме обычного прямоугольника класса со стереотипом «control».

Класс–сущность (entity) содержит информацию, которая должна храниться постоянно и не уничтожаться с уничтожением объектов данного класса или прекращением работы мо-

делируемой системы (выключением системы или завершением программы). Этот класс может соответствовать отдельной таблице базы данных, в этом случае его атрибуты могут быть полями таблицы, а операции — присоединенными процедурами. Как правило, этот класс является пассивным и лишь принимает сообщения от других классов модели. Класс–сущность может быть изображен также стандартным образом в форме обычного прямоугольника класса со стереотипом «entity».

Граничный класс (boundary) располагается на границе системы с внешней средой, но является составной частью системы. Граничный класс может быть изображен также стандартным образом в форме обычного прямоугольника класса со стереотипом «boundary».

Интерфейс (interface) в контексте языка UML является специальным случаем класса, у которого имеются только операции и отсутствуют атрибуты. Для обозначения интерфейса используется специальный графический символ: окружность или стандартный способ – прямоугольник класса со стереотипом «interface». На диаграмме вариантов использования интерфейс изображается в виде маленького круга, рядом с которым записывается его имя. Интерфейсы на диаграмме служат для спецификации таких элементов модели, которые видимы извне, но их внутренняя структура остается скрытой от клиентов. Применительно к диаграммам классов, интерфейсы определяют совокупность операций, которые обеспечивают необходимый набор возможных действий для актеров. С системно-аналитической точки зрения интерфейс определяет общие границы проектируемой системы. Графическое изображение интерфейсов в форме окружности может использоваться и на других типах канонических диаграмм, например, диаграммах развертывания и диаграммах вариантов использования.

Тема 8 Спецификация требований и рекомендации по написанию диаграммы классов.

Процесс разработки диаграммы классов занимает центральное место в ООАП сложных систем. От умения правильно выбрать классы и установить между ними взаимосвязи часто зависит не только успех процесса проектирования, но и производительность выполнения программы. Как показывает практика ООП, каждый программист в своей работе стремится в той или иной степени использовать уже накопленный личный опыт при разработке новых проектов. Это обусловлено желанием свести новую задачу к уже решенным, чтобы иметь возможность использовать не только проверенные фрагменты программного кода, но и отдельные компоненты в целом (библиотеки компонентов).

Такой стереотипный подход позволяет существенно сократить сроки реализации проекта, однако приемлем лишь в том случае, когда новый проект концептуально и технологически не слишком отличается от предыдущих. В противном случае платой за сокращение сроков проекта может стать его реализация на устаревшей технологической базе. Что касается собственно объектной структуризации предметной области, то здесь уместно придерживаться тех рекомендаций, которые накоплены в ООП. Они широко освещены в литературе [1, 2, 4, 10, 13, 18, 20] и поэтому здесь не рассматриваются.

При определении классов, атрибутов и операций и задании их имен и типов перед отечественными разработчиками всегда встает невольный вопрос: какой из языков использовать в качестве естественного, русский или английский? С одной стороны, использование родного языка для описания модели является наиболее естественным способом ее представления и в наибольшей степени отражает коммуникативную функцию модели системы. С другой стороны, разработка модели является лишь одним из этапов разработки соответствующей системы, а применение инструментальных средств для ее реализации в абсолютном большинстве случаев требует использования англоязычных терминов. Именно поэтому возникает характерная неоднозначность, с которой, по-видимому, совершенно незнакома англоязычная аудитория.

Отвечая на поставленный выше вопрос, следует отметить, что наиболее целесообразно придерживаться следующих рекомендаций. При построении диаграммы вариантов использования, являющейся наиболее общей концептуальной моделью проектируемой системы, применение русскоязычных терминов является не только оправданным с точки зрения описания структуры предметной области, но и эффективным с точки зрения коммуникативного взаимодействия с заказчиком и пользователями. При построении остальных типов диаграмм следует придерживаться разумного компромисса.

В частности, на начальных этапах разработки диаграмм целесообразность использования русскоязычных терминов вполне очевидна и оправдана. Однако, по мере готовности графической модели для реализации в виде программной системы и передачи ее для дальнейшей работы программистам, акцент может смещаться в сторону использования англоязычных терминов, которые в той или иной степени отражают особенности языка программирования, на котором предполагается реализация данной модели.

Более того, использование CASE-инструментариев для автоматизации ООАП, чаще всего, накладывает свои собственные требования на язык спецификации моделей. Именно по этой причине большинство примеров в литературе даются в англоязычном представлении, а при их переводе на русский может быть утрачена не только точность формулировок, но и семантика соответствующих понятий.

Пример диаграммы классов представлен на рис. 7

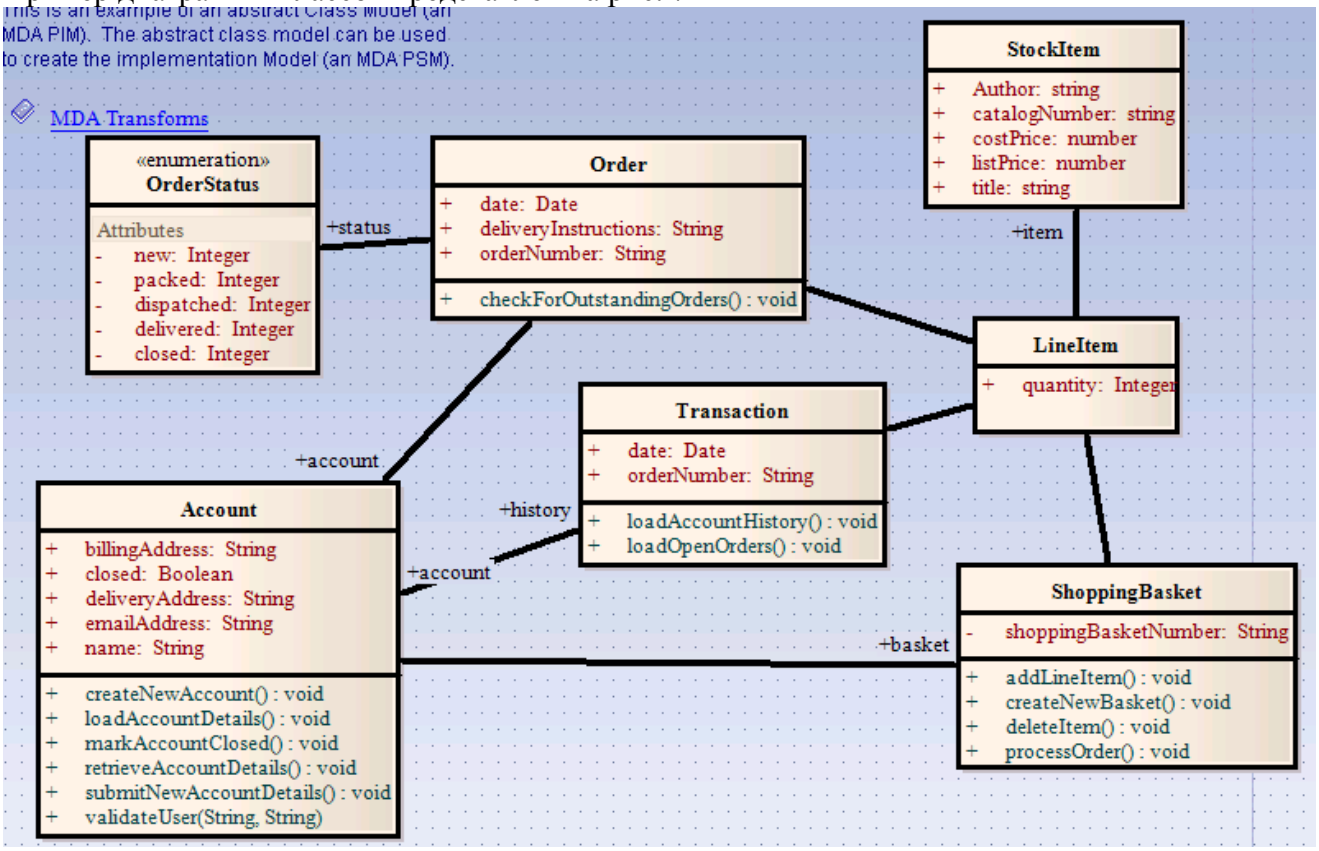

Рис. 7 Диаграмма классов

Тема 9 Элементы графической нотации диаграммы кооперации.

Понятие кооперации (collaboration) является одним из фундаментальных понятий в языке UML. Оно служит для обозначения множества взаимодействующих с определенной целью объектов в общем контексте моделируемой системы. Цель самой кооперации состоит в том, чтобы специфицировать особенности реализации отдельных наиболее значимых опе-

раций в системе. Кооперация определяет структуру поведения системы в терминах взаимодействия участников этой кооперации.

Кооперация может быть представлена на двух уровнях:

- На уровне спецификации — показывает роли классификаторов и роли ассоциаций в рассматриваемом взаимодействии.
- На уровне примеров — указывает экземпляры и связи, образующие отдельные роли в кооперации.

Диаграмма кооперации уровня спецификации показывает роли, которые играют участвующие во взаимодействии элементы. Элементами кооперации на этом уровне являются классы и ассоциации, которые обозначают отдельные роли классификаторов и ассоциации между участниками кооперации.

Диаграмма кооперации уровня примеров представляется совокупностью объектов (экземпляры классов) и связей (экземпляры ассоциаций). При этом связи дополняются стрелками сообщений. На данном уровне показываются только релевантные объекты, т. е. имеющие непосредственное отношение к реализации операции или классификатора.

В кооперации уровня примеров определяются свойства, которые должны иметь экземпляры для того, чтобы участвовать в кооперации. Кроме свойств объектов на диаграмме кооперации также указываются ассоциации, которые должны иметь место между объектами кооперации. При этом вовсе не обязательно изображать все свойства или все ассоциации, поскольку на диаграмме кооперации присутствуют только роли классификаторов, но не сами классификаторы. Таким образом, в то время как классификатор требует полного описания всех своих экземпляров, роль классификатора требует описания только тех свойств и ассоциаций, которые необходимы для участия в отдельной кооперации.

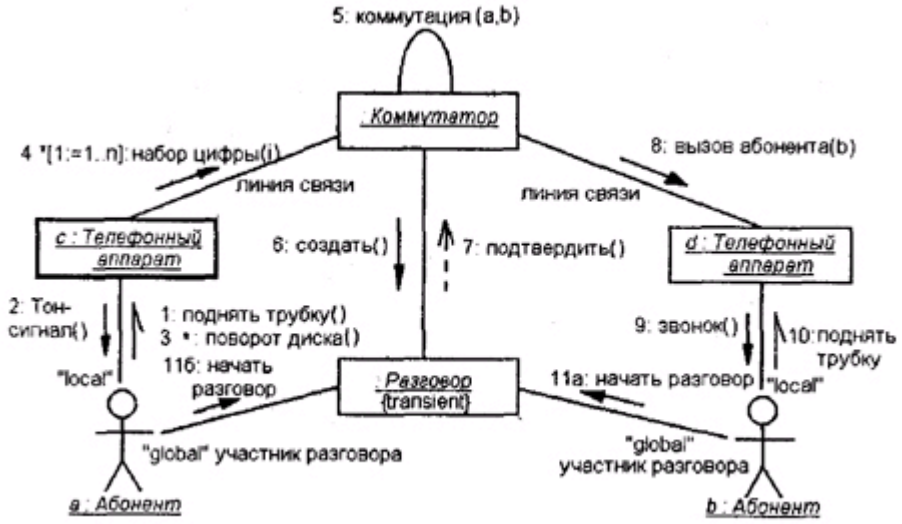

Рис. 8 Пример диаграммы кооперации

Тема 10 Элементы графической нотации диаграммы последовательности.

С помощью диаграммы последовательности описывают полный контекст взаимодействий между объектами для выполнения одного варианта использования.

На диаграмме последовательности изображаются объекты, которые непосредственно участвуют во взаимодействии, при этом никакие статические связи с другими объектами не визуализируются. Для диаграммы последовательности ключевым моментом является именно динамика взаимодействия объектов во времени. При этом диаграмма последовательности имеет как бы два измерения. Одно — слева направо - в виде вертикальных линий, каждая из

которых изображает линию жизни отдельного объекта, участвующего во взаимодействии.

При этом каждый объект графически изображается в форме прямоугольника и располагаете в верхней части своей линии жизни. Внутри прямоугольника записываются собственное имя объекта со строчной буквы и имя класса, разделенные двоеточием.

Крайним слева на диаграмме изображается объект, который является инициатором взаимодействия. Правее изображается другой объект, который непосредственно взаимодействует с первым. Таким образом, порядок расположения объектов на диаграмме последовательности определяется исключительно соображениями удобства визуализации их взаимодействия друг с другом.

Второе измерение диаграммы последовательности — вертикальная временная ось, направленная сверху вниз. Начальному моменту соответствует самая верхняя часть диаграммы. При этом процесс взаимодействия объектов реализуется посредством сообщений, которые посылаются одними объектами другим. Сообщения изображаются в виде горизонтальных стрелок с именем сообщения и образуют некоторый порядок относительно времени своей инициализации; сообщения, расположенные на диаграмме последовательности выше, передаются раньше тех, которые расположены ниже.

Линия жизни объекта (object lifeline) изображается пунктирной вертикальной линией. Линия жизни служит для обозначения периода, в течение которого объект существует в системе и, следовательно, может потенциально участвовать во всех ее взаимодействиях. Если объект существует в системе постоянно, то его линия жизни должна продолжаться по всей плоскости диаграммы последовательности от самой верхней ее части до самой нижней. С другой стороны, отдельные объекты, выполнив свою роль в системе, могут быть уничтожены (разрушены), чтобы освободить занимаемые ими ресурсы. Для таких объектов линия жизни обрывается в момент его уничтожения. Для обозначения момента уничтожения объекта в языке UML используется специальный символ в форме латинской буквы «X». Ниже этого символа пунктирная линия не изображается, поскольку соответствующего объекта в системе уже нет.

В процессе функционирования объектно-ориентированных систем одни объекты могут находиться в активном состоянии, непосредственно выполняя определенные действия или в состоянии пассивного ожидания сообщений от других объектов. Чтобы явно выделить подобную активность объектов, на диаграммах последовательности применяется специальное понятие, получившее название фокуса управления (focus of control). Фокус управления изображается в форме вытянутого узкого прямоугольника, верхняя сторона которого обозначает начало получения фокуса управления объекта (начало активности), а ее нижняя сторона — окончание фокуса управления (окончание активности).

С другой стороны, периоды активности объекта могут чередоваться с периодами его пассивности или ожидания. В этом случае у такого объекта фокусы управления изменяют свое изображение на линию жизни и наоборот.

В отдельных случаях инициатором взаимодействия в системе может быть актер или внешний пользователь. В этом случае актер изображается на диаграмме последовательности самым первым объектом слева со своим фокусом управления. Наиболее часто актер и его фокус управления будут существовать в системе постоянно, отмечая характерную для пользователя активность в инициировании взаимодействий с системой. При этом сам актер может иметь собственное имя либо оставаться анонимным.

На диаграммах последовательности могут присутствовать три разновидности сообщений. Первая разновидность сообщения является наиболее распространенной и используется для вызова процедур, выполнения операций или обозначения отдельных вложенных потоков управления. Начало этой стрелки, как правило, соприкасается с фокусом управления того объекта-клиента, который инициирует это сообщение. Конец стрелки соприкасается с линией жизни того объекта, который принимает это сообщение и выполняет в ответ определенные действия.

При этом принимающий объект получает фокус управления, становясь в этом случае

активным. Передающий объект может потерять фокус управления или остаться активным. Вторая разновидность сообщения используется для обозначения простого асинхронного сообщения, которое передается в произвольный момент.

Передача такого сообщения обычно не сопровождается получением фокуса управления объектом-получателем. Третья разновидность сообщения используется для возврата из вызова процедуры. Примером может служить простое сообщение о завершении некоторых вычислений без предоставления результата расчетов объекту-клиенту. В процедурных потоках управления эта стрелка может быть опущена, поскольку ее наличие неявно предполагается в конце активизации объекта. В то же время считается, что каждый вызов процедуры имеет свою пару — возврат вызова.

Таким образом, каждое сообщение ассоциируется с некоторым действием, которое должно быть выполнено принявшим его объектом. При этом действие может иметь некоторые аргументы или параметры, в зависимости от конкретных значений которых может быть получен различный результат. Соответствующие параметры будет иметь и вызывающее это действие сообщение. Более того, значения параметров отдельных сообщений могут содержать условные выражения, образуя ветвление или альтернативные пути основного потока управления.

Одной из особенностей диаграммы последовательности является возможность визуализировать простое ветвление процесса. Для изображения ветвления изображаются две или более стрелки, выходящие из одной точки фокуса управления объекта. При этом рядом с каждой из стрелок должно быть явно указано соответствующее условие ветви в форме булевского выражения. Хотя количество ветвей может быть произвольным, следует помнить, что наличие ветвлений может существенно усложнить интерпретацию диаграммы последовательности. Напомним, что предложение-условие может быть записано в форме обычного текста, псевдокода или выражения на языке программирования и должно представлять собой некоторое булевское выражение. Запись этих условий должна исключать одновременную передачу альтернативных сообщений по двум и более ветвям. В этом случае принято говорить об отсутствии конфликта для условий ветвления.

Сообщения могут иметь собственное имя, в качестве которого выступает имя операции, вызов которой инициируют эти сообщения у принимающего объекта. В этом случае рядом со стрелкой записывается имя операции с круглыми скобками, в которых могут указываться параметры или аргументы соответствующей операции. Если параметры отсутствуют, то скобки все равно должны быть изображены после имени операции. Примерами таких операций могут служить следующие: выдатьКлиентуНаличнымиСумму(n), установитьСоединениеМеждуАбонентами(a,b), сделатьВводимыйТекстНевидимым(), податьЗвуковойСигналТревоги().

В отдельных случаях выполнение тех или иных действий на диаграмме последовательности может потребовать явной спецификации временных ограничений, накладываемых на интервал выполнения операций или передачу сообщений. Для записи временных ограничений используются фигурные скобки. Временные ограничения записываются рядом с началом стрелки соответствующего сообщения. Если временное ограничение относится к конкретному объекту, то имя этого объекта записывается перед именем атрибута и отделяется от нее точкой.

Примерами таких ограничений на диаграмме последовательности могут служить ситуации, когда необходимо явно специфицировать время, в течение которого допускается передача сообщения от клиента к серверу или обработка запроса клиента сервером:

{времяОжиданияОтвета < 5 сек.}

{объект А. времяЗвучанияСигналаТревоги = 10 мин.}

Построение диаграммы последовательности целесообразно начинать с выделения из всей совокупности тех классов, объекты которых участвуют в моделируемом взаимодействии. После этого все объекты наносятся на диаграмму с соблюдением некоторого порядка инициализации сообщений. Здесь необходимо установить, какие объекты будут существо-

вать постоянно, а какие временно — только на период выполнения требуемых действий.

Когда объекты визуализированы, можно приступать к спецификации сообщений. При этом необходимо учитывать те роли, которые играют сообщения в системе. При необходимости уточнения этих ролей следует использовать их разновидности и стереотипы. Для уничтожения объектов, которые создаются на время выполнения своих действий, нужно предусмотреть явное сообщение.

Наиболее простые случаи ветвления процесса взаимодействия можно изобразить на одной диаграмме с использованием соответствующих графических примитивов. Однако следует помнить, что каждый альтернативный поток управления может существенно затруднить понимание построенной модели. Поэтому общим правилом является визуализация каждого потока управления на отдельной диаграмме последовательности. В этой ситуации такие отдельные диаграммы должны рассматриваться совместно как одна модель взаимодействия.

Дальнейшая детализация диаграммы последовательности связана с введением временных ограничений на выполнение отдельных действий в системе. Для простых асинхронных сообщений временные ограничения могут отсутствовать. Однако необходимость синхронизировать сложные потоки управления, как правило, требуют введения в модель таких ограничений.

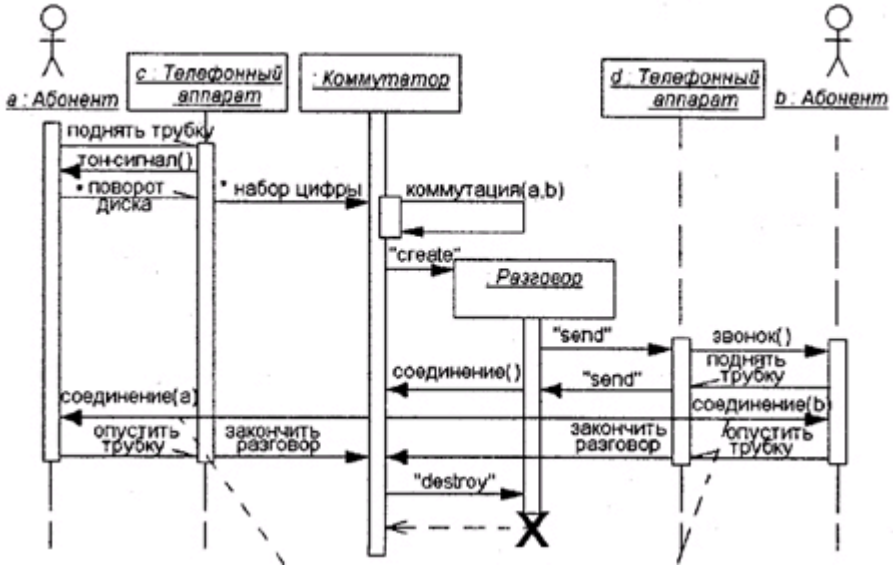

Рис. 9 Диаграмма последовательности

Тема 11 Элементы графической нотации диаграммы состояний

Главное предназначение этой диаграммы — описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение моделируемой системы. Диаграмма состояний представляет реакцию системы на конкретные события. Системы, которые реагируют на внешние действия от других систем или от пользователей, иногда называют реактивными.

Основными понятиями данной диаграммы являются состояние и переход. Основное различие между ними заключается в том, что длительность нахождения системы в отдельном состоянии существенно превышает время, которое затрачивается на переход из одного состояния в другое. Моделируемая система представляется в виде ориентированного графа, вершины которого соответствуют состояниям, а дуги — переходам. При этом поведение представляет последовательное перемещение по графу состояний от вершины к вершине, по связывающим их дугам с учетом их ориентации от начального до конечного состояний.

Допускается вложение одних состояний в другие для уточнения внутренней структу-

ры отдельных более общих состояний (макросостояний).

Формализм состояний основан на выполнении следующих обязательных условий.

Не учитывается последовательность перемещения из состояния в состояние. Определяющим является сам факт нахождения моделируемого элемента в том или ином состоянии, но не последовательность состояний, в результате которой элемент перешел в текущее состояние.

В каждый момент система может находиться только в одном из своих состояний. При этом система может находиться в отдельном состоянии как угодно долго, если не происходит никаких событий.

Количество состояний должно быть обязательно конечным, и все они должны быть специфицированы явным образом.

Диаграмма не должна содержать изолированных состояний и переходов. Это условие означает, что для каждого из состояний, кроме начального, должно быть определено хотя бы одно предшествующее состояние. Каждый переход должен обязательно соединять два состояния.

Не должно быть конфликтующих переходов, т. е. таких переходов из одного и того же состояния, когда при наступлении одного и того же события моделируемый элемент одновременно может перейти в два и более последующих состояния (кроме случая параллельных конечных подавтоматов). Исключение конфликтов возможно на основе введения сторожевых условий.

Состояние может быть задано в виде набора конкретных значений атрибутов объекта некоторого класса, при этом изменение отдельных значений данных атрибутов будет отражать изменение состояния моделируемого объекта или системы в целом. Следует заметить, что не каждый атрибут класса может характеризовать состояние его объектов. Как правило, имеют значение только такие свойства элементов системы, которые отражают динамический или функциональный аспект ее поведения. В этом случае состояние будет характеризоваться условием, включающим в себя только значимые для поведения объекта или системы атрибуты классов и их значения.

Например, такое условие может представлять статическую ситуацию, когда объект находится в состоянии ожидания возникновения некоторого внешнего события. С другой стороны, аналогичное условие используется для моделирования динамических аспектов, когда в ходе нахождения объекта в некотором состоянии выполняются некоторые действия. В последнем случае соответствующая деятельность начинается в момент перехода моделируемого элемента в рассматриваемое состояние, и элемент может покинуть данное состояние в момент завершения этой деятельности.

Состояние на диаграмме изображается прямоугольником с закругленными вершинами. Этот прямоугольник, в свою очередь, может быть разделен на две секции горизонтальной линией. Если указана лишь одна секция, то в ней записывается только имя состояния. В противном случае в первой из них записывается имя состояния, а во второй — список некоторых внутренних действий или переходов в данном состоянии.

Имя состояния должно представлять собой законченное предложение и всегда записываться с заглавной буквы. Поскольку состояние системы является составной частью процесса ее функционирования, рекомендуется в качестве имени использовать глаголы в настоящем времени (звенит, печатает, ожидает).

Для некоторых состояний может потребоваться дополнительно указать некоторые действия, которые должны быть выполнены моделируемым элементом при нахождении его в том или ином состоянии. Для этой цели служит дополнительная секция в обозначении состояния или примечание, содержащая перечень внутренних действий или деятельность, которые выполняются в процессе нахождения моделируемого элемента в данном состоянии. Каждое из действий записывается в виде отдельной строки и имеет следующий формат:

<метка действия '/ ' выражение действия>

Перечень меток действий в языке UML фиксирован:

entry — указывает на то, что следующее за ней выражение действия должно быть выполнено в момент входа в данное состояние (входное действие);

exit — указывает на то, что следующее за ней выражение действия должно быть выполнено в момент выхода из данного состояния (выходное действие);

do — специфицирует некоторую деятельность (do activity) или так называемую дудеятельность, которая выполняется в течение всего времени, пока объект находится в данном состоянии, или до тех пор, пока не будет выполнено условие ее окончания, специфицированное в соответствующей операции класса или вычислительной процедуре. В последнем случае при завершении деятельности генерируется соответствующее событие;

include — используется для обращения к подсостоянию, при этом следующее за ней выражение действия содержит имя этого подсостояния.

Начальное состояние (initial state) представляет собой частный случай состояния, которое не содержит никаких внутренних действий. Оно служит для указания на диаграмме состояний графической области, от которой начинается процесс изменения состояний. Графически начальное состояние обозначается в виде закрашенного кружка, из которого может только выходить стрелка-переход. На самом верхнем уровне представления объекта переход из начального состояния может быть помечен событием создания (инициализации) данного объекта. В противном случае этот переход никак не помечается.

Конечное состояние (final state) представляет собой частный случай состояния, которое также не содержит никаких внутренних действий. В этом состоянии должен находиться моделируемый объект или система по умолчанию после завершения работы.

Простой переход (simple transition) представляет собой отношение между двумя последовательными состояниями, которое указывает на факт смены одного состояния другим. Пребывание моделируемого объекта или системы в первом состоянии может сопровождаться выполнением некоторых внутренних действий (деятельности), при этом переход в другое состояние будет возможен либо после завершения этих действий (деятельности), либо при возникновении некоторых событий.

Переход осуществляется при наступлении некоторого события: окончания выполнения деятельности (do activity), получения объектом сообщения или приемом сигнала. На переходе указывается имя события. Кроме того, на переходе могут указываться действия, производимые объектом в ответ на внешние события при переходе из одного состояния в другое. Срабатывание перехода может зависеть не только от наступления некоторого события, но и от выполнения определенного условия, называемого сторожевым условием. Объект перейдет из одного состояния в другое в том случае, если произошло указанное событие и сторожевое условие приняло значение «истина».

На диаграмме состояний переход изображается сплошной линией со стрелкой, которая выходит из исходного состояния и направлена в целевое состояние. Каждый переход может быть помечен строкой текста, которая имеет следующий общий формат:

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

При этом сигнатура события описывает некоторое событие с необходимыми аргументами: <имя события>'('<список параметров, разделенных запятыми>')'.

Термин событие (event) представляет собой спецификацию некоторого факта, имеющего место в пространстве и во времени. Про события говорят, что они «происходят», при этом отдельные события должны быть упорядочены во времени. После наступления некоторого события нельзя уже вернуться к предыдущим событиям, если такая возможность не предусмотрена явно в модели.

События стимулируют переходы из одних состояний в другие. В качестве событий можно рассматривать сигналы, вызовы, окончание фиксированных промежутков времени или моменты окончания выполнения определенных действий. В зависимости от вида происходящих событий-стимулов различают два типа переходов: триггерные и нетриггерные.

Переход называется триггерным, если его специфицирует некоторое событие-триггер. В этом случае рядом со стрелкой триггерного перехода обязательно указывается

имя события в форме строки текста, начинающейся со строчной буквы. Наиболее часто в качестве имен триггерных переходов задают имена операций, вызываемых у тех или иных объектов системы. После имени такого события следуют круглые скобки для явного задания параметров соответствующей операции. Если таких параметров нет, то список со скобками может отсутствовать.

Переход называется нетриггерным, если он происходит по завершении выполнения действий (деятельности) в исходном состоянии. Нетриггерные переходы часто называют переходами по завершении ду-деятельности. Для них рядом со стрелкой перехода не указывается никакого имени события, а в исходном состоянии должна быть описана внутренняя ду-деятельность, по завершении которой произойдет тот или иной нетриггерный переход.

Сторожевое условие (guard condition), если оно есть, всегда записывается в прямых скобках и представляет собой некоторое булевское выражение.

Если сторожевое условие принимает значение «истина», то соответствующий переход при наступлении события-триггера или завершения деятельности может сработать, в результате чего объект перейдет в новое состояние. Если сторожевое условие принимает значение «ложь», то переход не может сработать, даже если произошло событие-триггер или завершилась деятельность в исходном переходе. Очевидно, в случае невыполнения сторожевого условия моделируемый объект или система останется в исходном состоянии.

Поскольку общее количество выходящих из состояния переходов в языке UML никак не ограничено (но конечно), то не исключена ситуация, когда из одного состояния могут выходить несколько переходов с одним и тем же событием-триггером. Каждый из таких переходов должен содержать собственное сторожевое условие, но при этом никакие два или более сторожевых условий не должны одновременно принимать значение «истина». В противном случае на диаграмме состояний будет иметь место конфликт триггерных переходов, что делает несостоятельной (ill formed) модель системы в целом.

Выражение действия (action expression) выполняется только в том случае, когда переход срабатывает. Представляет собой вызов операции или передачу некоторого сообщения, имеет атомарный характер и выполняется сразу после срабатывания соответствующего перехода до начала каких бы то ни было действий в целевом состоянии. Атомарность действия означает, что оно не может быть прервано никаким другим действием до тех пор, пока не закончится его выполнение.

В общем случае выражение действия может содержать целый список отдельных действий, разделенных символом «;». Обязательное требование –все действия из списка должны четко различаться между собой и следовать в порядке их записи. На синтаксис записи выражений действия не накладывается никаких ограничений.

Составное состояние (composite state) — такое сложное состояние, которое состоит из других вложенных в него состояний. Последние выступают по отношению к первому как подсостояния (substate). Хотя между ними имеет место отношение композиции, графически все вершины диаграммы, которые соответствуют вложенным состояниям, изображаются внутри символа составного состояния. В этом случае размеры графического символа составного состояния увеличиваются так, чтобы вместить в себя все подсостояния. При этом любое из подсостояний, в свою очередь, может являться составным состоянием и содержать внутри себя другие вложенные подсостояния. Количество уровней вложенности составных состояний в языке UML не фиксировано.

Последовательные подсостояния (sequential substates) используются для моделирования такого поведения объекта, во время которого в каждый момент объект может находиться в одном и только одном подсостоянии. Поведение объекта в этом случае представляет собой последовательную смену подсостояний, начиная от начального и заканчивая конечным подсостоянием. Хотя моделируемый объект или система продолжает находиться в составном состоянии, введение в рассмотрение последовательных подсостояний позволяет учесть более тонкие логические аспекты его внутреннего поведения.

Параллельные подсостояния (concurrent substates) позволяют моделировать состояния,

которые могут выполняться параллельно внутри составного события.

В отдельных случаях возникает необходимость явно показать ситуацию, когда некоторый переход может иметь несколько исходных состояний или несколько целевых состояний. Такой переход получил специальное название — параллельный переход. Введение в рассмотрение параллельных переходов может быть обусловлено необходимостью синхронизировать и/или разделить отдельные процессы управления на параллельные нити без спецификации дополнительной синхронизации в параллельных конечных подавтоматах. Графически такой переход изображается вертикальной или горизонтальной черточкой. Если параллельный переход имеет две или более входящие дуги, то его называют слиянием (join). Если же он имеет две или более исходящих из него дуг, то его называют разделением (fork). Текстовая строка спецификации параллельного перехода записывается рядом с черточкой и относится ко всем входящим (исходящим) дугам.

Срабатывание параллельного перехода происходит следующим образом. В первом случае переход-слияние срабатывает, если имеет место событие-триггер для всех исходных состояний этого перехода и выполнено (при его наличии) сторожевое условие.

Во втором случае происходит и образуются параллельные ветви вложенных подпроцессов. При этом после срабатывания перехода-разделения моделируемая система или объект одновременно будет находиться во всех целевых подсостояниях этого параллельного перехода. Далее процесс изменения состояний будет протекать согласно ранее рассмотренным правилам для составных состояний.

При выделении состояний и переходов следует помнить, что длительность срабатывания отдельных переходов должна быть существенно меньше, чем нахождение моделируемых элементов в соответствующих состояниях. Каждое из состояний должно характеризоваться определенной устойчивостью во времени. Все переходы должны быть явно специфицированы, в противном случае построенная диаграмма состояний является либо неполной (неадекватной), либо ошибочной с точки зрения нотации языка UML (ill formed).

При разработке диаграммы состояний нужно постоянно следить, чтобы объект в каждый момент находился только в единственном состоянии.

Следует выполнять обязательную проверку того, чтобы никакие два перехода из одного состояния не могли сработать одновременно (требование отсутствия конфликтов у переходов). Если параллельность по замыслу разработчика отсутствует, то следует ввести дополнительные сторожевые условия либо изменить существующие, чтобы исключить конфликт переходов. При наличии параллельности следует заменить конфликтующие переходы одним параллельным переходом типа ветвления.

Фрагмент диаграммы состояний АСОИ представлен на рисунке 10.

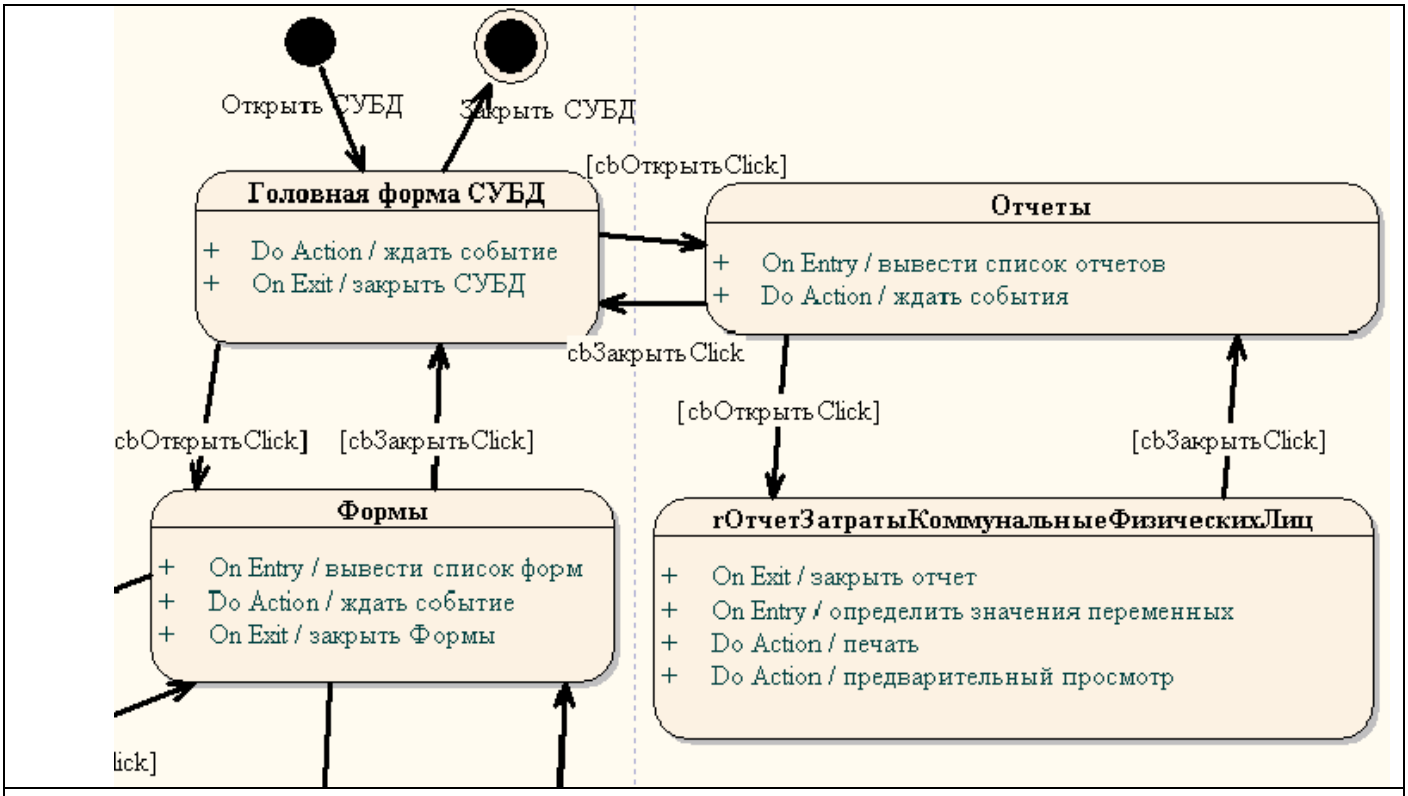

Рисунок10 –Диаграмма состояний АСОИ

Тема 12 Элементы графической нотации диаграммы деятельности.

При моделировании поведения проектируемой или анализируемой системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической и процедурной реализации выполняемых системой операций. Традиционно для этой цели использовались блок-схемы или структурные схемы алгоритмов.

Для моделирования процесса выполнения операций используются диаграммы деятельности. Применяемая в них графическая нотация во многом похожа на нотацию диаграммы состояний, поскольку на диаграммах деятельности также присутствуют обозначения состояний и переходов. Отличие заключается в отсутствии на переходах сигнатуры событий. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние срабатывает только при завершении операции в предыдущем состоянии.

В контексте языка UML деятельность (activity) представляет собой некоторую совокупность отдельных вычислений. При этом отдельные элементарные вычисления могут приводить к некоторому результату или действию (action). На диаграмме деятельности отображается логика или последовательность перехода от одной деятельности к другой, при этом внимание фиксируется на результате деятельности.

Состояние действия (action state) является специальным случаем состояния с некоторым входным действием и, по крайней мере, одним выходящим из состояния переходом. Обычное использование состояния действия заключается в моделировании одного шага выполнения алгоритма (процедуры) или потока управления.

Графически состояние действия изображается фигурой, напоминающей прямоугольник со сферическими боковыми сторонами. Внутри этой фигуры записывается имя состояния действия в форме выражение действия (action-expression), которое должно быть уникальным в пределах одной диаграммы деятельности.

Действие может быть записано на естественном языке, некотором псевдокоде или

языке программирования. Никаких дополнительных или неявных ограничений при записи действий не накладывается. Рекомендуется в качестве имени простого действия использовать глагол с пояснительными словами. Если же действие может быть представлено в некотором формальном виде, то целесообразно записать его на том языке программирования, на котором предполагается реализовывать разрабатываемый проект.

Иногда возникает необходимость представить на диаграмме деятельности некоторое сложное действие, состоящее из нескольких более простых. В этом случае можно использовать специальное обозначение так называемого состояния поддеятельности (subactivity state). Это состояние является графом деятельности и обозначается специальной пиктограммой в правом нижнем углу символа состояния действия. Данная конструкция может применяться к любому элементу языка UML, который поддерживает «вложенность» своей структуры. При этом пиктограмма может быть дополнительно помечена типом вложенной структуры.

Каждая диаграмма деятельности должна иметь единственное начальное и конечное состояния. Они имеют такие же обозначения, как и на диаграмме состояний. При этом каждая деятельность начинается в начальном состоянии и заканчивается в конечном состоянии. Саму диаграмму деятельности принято располагать таким образом, чтобы действия следовали сверху вниз. В этом случае начальное состояние будет изображаться в верхней части диаграммы, а конечное — в ее нижней части.

При построении диаграммы деятельности используются только нетриггерные переходы, т. е. те, которые срабатывают сразу после завершения деятельности или выполнения соответствующего действия состояния. На диаграмме такой переход изображается сплошной линией со стрелкой.

Если из состояния действия выходит единственный переход, то он может быть никак не помечен. Если же таких переходов несколько, при моделировании последовательной деятельности может сработать только один из них. В этом случае для каждого из таких переходов должно быть явно записано собственное сторожевое условие в прямых скобках. При этом для всех выходящих из некоторого состояния переходов должно выполняться требование истинности только одного из них.

Графически ветвление на диаграмме деятельности обозначается символом решения (decision), изображаемого в форме небольшого ромба, внутри которого нет никакого текста. В этот ромб может входить только одна стрелка от того состояния действия, после выполнения которого поток управления должен быть продолжен по одной из взаимно исключающих ветвей. Принято входящую стрелку присоединять к верхней или левой вершине символа решения. Выходящих стрелок может быть две или более, но для каждой из них явно указывается соответствующее сторожевое условие в форме булевского выражения.

Один из наиболее значимых недостатков обычных блок-схем или структурных схем алгоритмов связан с проблемой изображения параллельных ветвей отдельных вычислений. Поскольку распараллеливание вычислений существенно повышает общее быстродействие программных систем, необходимы графические примитивы для представления параллельных процессов. Для диаграмм деятельности для этой цели используется специальный символ для разделения и слияния параллельных вычислений или потоков управления. Таким символом является прямая черточка, выступающая аналогично обозначению параллельных переходов для диаграмм состояний.

На диаграммах деятельности такая черточка изображается отрезком горизонтальной, реже — вертикальной линии, толщина которой несколько шире линий простых переходов диаграммы деятельности. При этом разделение (concurrent fork) имеет один входящий переход и несколько выходящих, которые изображаются отрезками вертикальных, реже — горизонтальных линий. Слияние (concurrent join), наоборот, имеет несколько входящих переходов и один выходящий. Особенностью параллельных переходов на диаграмме деятельности является возможность их изображения в удлиненной форме для возможности показать входящие и выходящие переходы вертикальными стрелками.

Напомним, что базовым графическим представлением объекта в нотации языка UML

является прямоугольник класса, с тем отличием, что имя объекта подчеркивается. На диаграммах деятельности после имени может указываться характеристика состояния объекта в прямых скобках. Такие прямоугольники объектов присоединяются к состояниям действия отношением зависимости пунктирной линией со стрелкой. Соответствующая зависимость определяет состояние конкретного объекта после выполнения предшествующего действия.

Диаграммы деятельности играют важную роль в понимании процессов реализации алгоритмов выполнения операций классов и процедурных потоков управления в моделируемой системе. Используемые для этой цели традиционные блок-схемы алгоритмов обладают серьезными ограничениями в представлении параллельных процессов и их синхронизации. Применение дорожек и объектов открывает дополнительные возможности для наглядного представления бизнес-процессов, позволяя специфицировать деятельность подразделений коммерческих компаний и фирм.

Содержание диаграммы деятельности во многом напоминает диаграмму состояний, хотя и не тождественно ей. Поэтому многие рекомендации по построению последней оказываются справедливыми применительно к диаграмме деятельности. В частности, диаграмма деятельности может быть построена для объектов отдельных классов, варианта использования, отдельной операции класса или моделируемой системы в целом.

С одной стороны, на начальных этапах проектирования, когда детали реализации деятельности в проектируемой системе неизвестны, построение диаграммы деятельности начинают с выделения поддеятельностей, которые в совокупности образуют исходную деятельность системы. В последующем, по мере разработки диаграмм классов и состояний, эти поддеятельности уточняются в виде отдельных вложенных диаграмм деятельности для подсистем и объектов.

С другой стороны, отдельные участки рабочего процесса в существующей системе могут быть хорошо отлаженными, и у разработчиков может возникнуть желание сохранить этот механизм выполнения действий в проектируемой системе. Тогда строится диаграмма деятельности для этих участков, отражающая конкретные особенности выполнения действий с использованием дорожек и потока объектов. В последующем такая диаграмма вкладывается в более общие диаграммы деятельности для подсистемы и системы в целом, сохраняя свой уровень детализации.

В случае типового проекта большинство деталей реализации действий могут быть известны заранее на основе анализа существующих систем или предшествующего опыта разработки систем-прототипов. Для этой ситуации доминирующим будет восходящий процесс разработки. Использование типовых решений может существенно сократить время разработки и избежать возможных ошибок при реализации проекта.

При разработке проекта новой системы, процесс функционирования которой основан на новых технологических решениях, ситуация представляется более сложной. А именно: до начала работы над проектом могут быть неизвестны не только детали реализации отдельных действий, но и само содержание деятельности становится предметом разработки. В этом случае доминирующим будет нисходящий процесс разработки от более общих схем к уточняющим их диаграммам. При этом достижение такого уровня детализации всех диаграмм, который достаточен для понимания особенностей реализации всех действий, может служить признаком завершения отдельных этапов работы над проектом. Диаграммы деятельности должны быть представлены для всех объектов АСОИ, в которых производятся вычисления

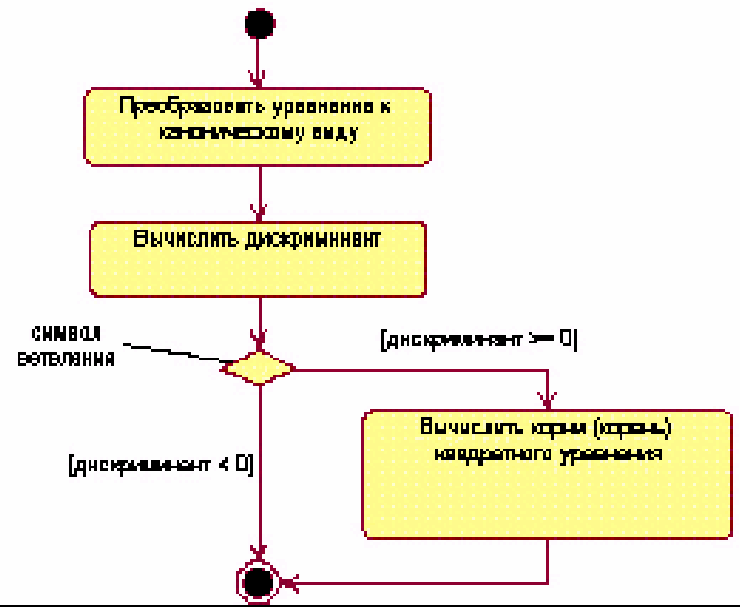

Рис. 11 Пример диаграммы деятельности

Тема 13 Элементы графической нотации диаграммы компонентов

Диаграммы компонентов показывают, как выглядит модель на физическом уровне. На них изображены компоненты программного обеспечения и связи между ними. При этом на такой диаграмме выделяют два типа компонентов: исполняемые компоненты и библиотеки кода. Каждый класс модели (или подсистема) преобразуется в компонент исходного кода. После создания они сразу добавляются к диаграмме компонентов. Между отдельными компонентами изображают зависимости, соответствующие зависимостям на этапе компиляции или выполнения программы. Если система разрабатывается на языке C++, то у каждого класса имеется свой собственный заголовочный файл и файл с расширением .CPP, так что каждый класс преобразуется в свои собственные компоненты на диаграмме.

Число диаграмм компонентов зависит от числа подсистем или исполняемых файлов. Каждая подсистема является пакетом компонентов. В общем случае пакеты – это совокупности компонентов.

Диаграммы компонентов применяются теми участниками проекта, кто отвечает за компиляцию системы. Из нее видно, в каком порядке надо компилировать компоненты, а также какие исполняемые компоненты будут созданы системой. На такой диаграмме показано соответствие классов реализованным компонентам. Она нужна там, где начинается генерация кода.

В нашем примере диаграмма компонентов помогает уяснить необходимое операционное и прикладное обеспечение АСОИ.

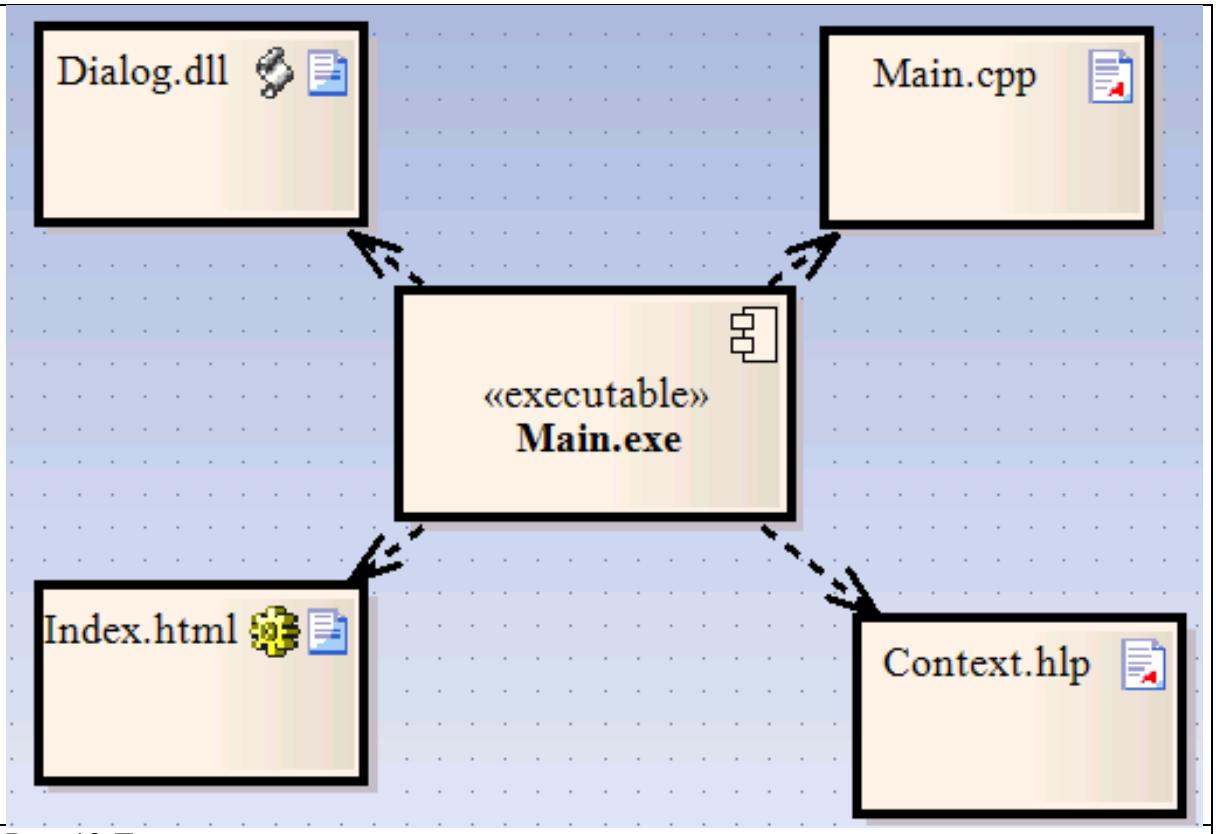

Рис. 12 Диаграмма компонентов.

Тема 14 Элементы графической нотации диаграммы развертывания

Диаграмма размещения (deployment diagram) отражает физические взаимосвязи между программными и аппаратными компонентами системы. Она является хорошим средством для того, чтобы показать маршруты перемещения объектов и компонентов в распределенной системе. Каждый узел на диаграмме размещения представляет собой некоторый тип вычислительного устройства – в большинстве случаев часть аппаратуры. Эта аппаратура может быть простым устройством или датчиком. Диаграмма размещения показывает физическое расположение сети и местонахождение в ней различных компонентов. В нашем примере система размещается на одном рабочем месте. Диаграмма размещения представлена на рисунке 13.

Диаграмма размещения используется менеджером проекта, пользователями, архитектором системы и эксплуатационным персоналом, чтобы понять физическое размещение системы и расположение её отдельных подсистем.

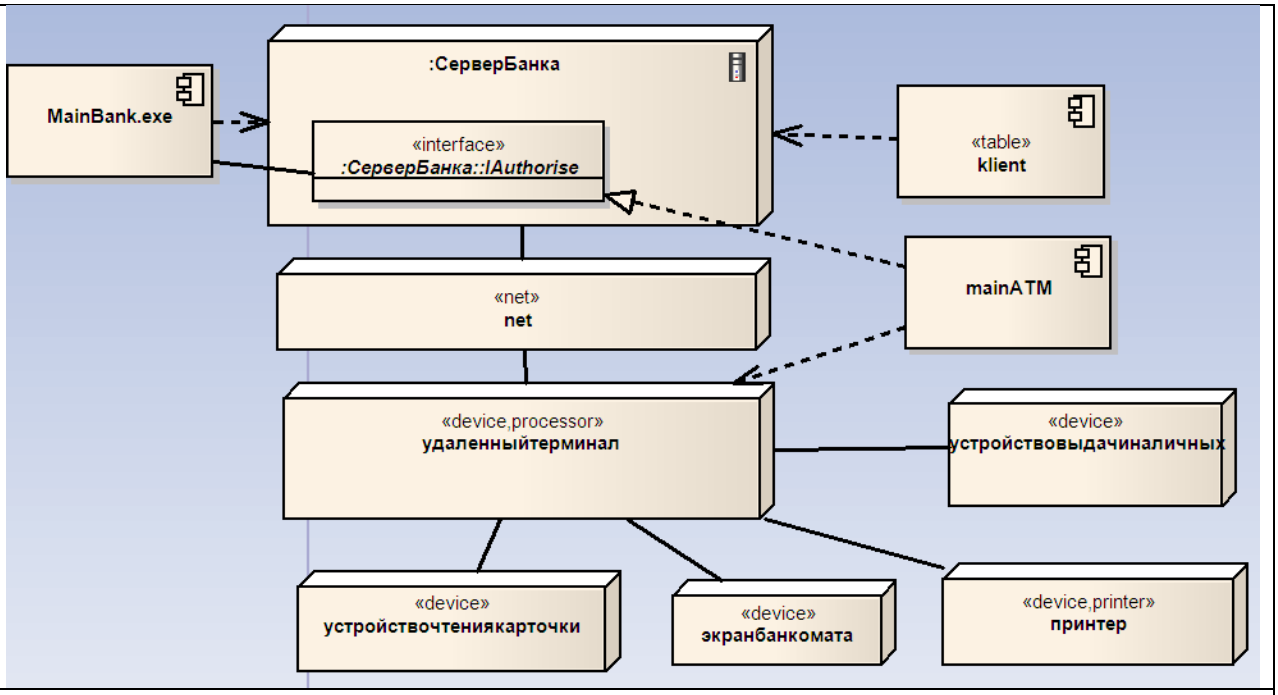

Рис. 13 Диаграмма размещения

Тема 15 Обеспечение защиты и достоверности информации

Основные угрозы информационной безопасности Понятие "информационная безопасность" несколько шире понятия "защита информации". т.к. под информационной безопасностью понимается защищенность не только информации, но и поддерживающей ее инфраструктуры от случайных или преднамеренных воздействий (искусственного или естественного характера) которые могут нанести ущерб владельцам или пользователям информации или поддерживающей ее инфраструктуры. Нам импонирует подход к обеспечению информационной безопасности, состоящий в том, что надо начинать с выявления субъектов информационных отношений, их целей, интересов и возможностей использования разного рода ресурсов, как для защиты своей информации, так и для нанесения ущерба другим субъектам и их информационным системам. В последнее время появились работы прямо употребляющие термин "информационная война". Понятие "информационная война" часто используют, как указывается в неправильно и указывают только на применение высоких информационных технологий в обычных вооруженных силах. Определение информационной войны : Действия, предпринятые для достижения информационного превосходства путем нанесения ущерба информации, процессам, основанным на информации, и информационным системам противника при одновременной защитесобственной информации, процессов, основанных на информации, и информационных систем. Понятие информационной войны с некоторыми оговорками уместно применить и взаимодействию субъектов в бизнесе, если интересы этих субъектов существенно различаются Проблему обеспечения защиты информации часто сужают до проблемы обеспечения защиты только компьютерной информации. Для комплексного подхода необходимо рассмотреть ряд аспектов информационной безопасности, в частности обеспечение разведзащищенности объекта. Под разведзащищенностью объекта предлагается понимать способность объекта противостоять всем видам разведки, то есть обеспечение разведзащищенности заключается в реализации комплекса организационных и технических мероприятий, направленных на обеспечение безопасности информации о самом объекте. В основном имеется в виду снижение информативности данных, "демаскирующих" объект, обрабатывая которые, злоумышленник может по-

лучить некоторую информацию о режиме функционирования объекта, его взаимодействию с другими участниками процесса. Здесь уместно отметить, что в последнее время большое внимание уделяется новому направлению в информационной безопасности, а именно "стеганографии", и в частности компьютерной стеганографии. Слово "стеганография" происходит от греческих слов steganos (секрет, тайна) и graphy (запись) и, таким образом, означает буквально "тайнопись". Когда в V веке до н.э. тиран Гистий, находясь под надзором царя Дария в Сузах, должен был послать секретное сообщение своему родственнику в анатолийский город Милет, он побрил наголо своего раба и вытатуировал послание на его голове. Когда волосы снова отросли, раб отправился в путь. Так Геродот описывает один из первых случаев применения в древнем мире стеганографии - искусства скрытого письма. Искусство развивалось, превратившись в науку, помогавшую людям на протяжении многих веков скрывать от посторонних глаз сам факт передачи информации. Еще древние римляне писали между строк невидимыми чернилами, в качестве которых использовались фруктовые соки, молоко и некоторых другие натуральные вещества. Компьютерные технологии придали новый импульс развитию и совершенствованию стеганографии, появилось новое направление в области защиты информации - компьютерная стеганография. Современный прогресс в области глобальных компьютерных сетей и средств мультимедиа привел к разработке новых методов, предназначенных для обеспечения безопасности передачи данных по каналам телекоммуникаций и использования их в необъявленных целях. Эти методы, учитывая естественные неточности устройств оцифровки и избыточность аналогового видео или аудио сигнала, позволяют скрывать сообщения в компьютерных файлах (контейнерах). Причем, в отличие от криптографии, данные методы скрывают сам факт передачи информации. Как следует из практики работы компьютерных систем видно, при использовании новых информационных технологий угрозам, в первую очередь подвергаются такие свойства информации, как конфиденциальность, целостность, доступность.

Будем считать, что опасными факторами или опасными воздействующими факторами (ОВФ) на защищаемые информационные ресурсы объекта являются: В части несанкционированного доступа (НСД) к защищаемым информационным ресурсам:

* различные версии (одной и той же программы) - угроза: целостности;
* ошибки программирования - угроза: конфиденциальности, целостности, доступности;
* умышленное повреждение данных и программ - угроза: целостности;
* потайные ходы и лазейки - угроза: конфиденциальности, целостности, доступности;
* вредоносные программы - угроза: целостности и доступности, могут затрагивать и конфиденциальность;
* аппаратные сбои - угроза: конфиденциальности, целостности и доступности;
* неточная или устаревшая информация - угроза: целостности;
* небрежность - угроза: конфиденциальности, целостности, доступности;
* искажение информации - угроза: целостности;
* кража (хищение) - угроза: конфиденциальности, целостности, доступности;
* мошенничество - угроза: целостности;
* подлог - угроза: целостности и другим ресурсам;
* самозванство - угроза: конфиденциальности, целостности, доступности;
* применение сетевых анализаторов - угроза: конфиденциальности.

В части специальных программно-математических воздействий или "вредоносных программ" (ВрП); в том числе через специально внедренные электронные и программные "закладки".

В части утечки информации по техническим каналам объекта:

* побочные электромагнитные излучения информативного сигнала от объекта информатизации;

* радиоизлучения, модулированные информативным сигналом, возникающие при работе различных генераторов, входящих в состав объекта, или при наличии паразитной генерации в узлах (элементах) этих средств; из помещений и зданий объекта:

* акустическое излучение информативного речевого сигнала или сигнала, обусловленного функционированием технических средств обработки информации (телеграф, телетайп, принтер, пишущая машинка и т.п.);

* электрические сигналы, возникающие посредством преобразования информативного сигнала из акустического в электрический за счет микрофонного эффекта, и распространение их по проводам и линиям передачи информации;

* вибрационные сигналы, возникающие посредством преобразования информативного сигнала из акустического в вибрационный при воздействии его на строительные конструкции и инженерно-технические коммуникации выделенных помещений;

* электрические сигналы или радиоизлучения, обусловленные воздействием на технические средства (ТС) высокочастотных сигналов, создаваемых с помощью разведывательной аппаратуры, по эфиру и проводам, либо сигналов промышленных радиотехнических устройств (радиовещательные, радиолокационные станции, средства радиосвязи и т.п.), модуляцией их информативным сигналом (облучение, "навязывание");

* радиоизлучения или электрические сигналы от внедренных в ТС и помещения (здания) объекта специальных электронных закладных устройств (СЭЗУ - "закладок") для перехвата информации, модулированные информативным сигналом; Опасные воздействующие факторы (ОВФ) и угрозы от них не имеют четко выявленной природы и, как правило, угрожают конфиденциальности, целостности, доступности информации, материальным ценностям и др. ресурсам и носят противозаконный или аморальный характер.

Проблема защиты произвольного объекта информатизации (ОИ) имеет два взаимосвязанных аспекта:

* аналитический (соответствующий "нападению" или "вскрытию систем");

* синтетический (защита).

Рассматриваемая проблема содержит компоненты:

* кто производит доступ (субъект);

* к чему производится доступ (объект);

* что есть "доступ"; * и по какому критерию "санкционированный" или легальный доступ отличается от "несанкционированного".

Основными источниками опасных факторов являются: деятельность человека, непосредственно и опосредовано влияющая на информационную безопасность объекта ; случайные источники (халатность, некомпетентность и пр.). стихийные бедствия и катастрофы. Умышленная деятельность человека: деятельность иностранных разведывательных и специальных служб по добыванию информации, навязыванию ложной информации, нарушению работоспособности ОИ и его отдельных компонент; противозаконная и преступная деятельность, направленная против интересов предприятия (промышленный шпионаж, недобросовестная конкурентная борьба, такие события - как пожары, взрывы, технические аварии и т.д.); нарушения пользователями и обслуживающим персоналом ОИ установленных регламентов сбора, обработки и передачи информации, а также требований информационной безопасности.

Непреднамеренные действия человека по отношению к информации и ОИ:

* некомпетентные действия и ошибки, допущенные при проектировании ОИ и его системы защиты информации;

* непреднамеренные ошибки пользователей и обслуживающего персонала ОИ, в том числе администратора службы БИ;

* халатность и недостаточно четкое исполнение служебных обязанностей. Отказы и неисправности средств информатизации: отказы и неисправности технических средств обработки, хранения и передачи информации; отказы и неисправности средств защиты информации и средств контроля; ошибки в разработке программных средств; сбои программ-

ного обеспечения. События, действия или явления, которые могут вызвать нарушение безопасности информации на объекте информатизации, называют угрозами. Источники угроз информационной безопасности делятся на внешние и внутренние:

К внешним источникам относятся: деятельность, направленная на неправомерный доступ к информации (на машинных носителях, в ЭВМ, сетях на их базе; распространение искажённой информации о предприятии; деятельность определенных структур, искажающих или скрывающих информацию, направленная против интересов государства и предприятия; действия преступных групп и отдельных лиц по добыванию информации с целью реализации своих замыслов;

Внутренними источниками угроз являются: ввод в эксплуатацию объектов информатизации, не отвечающих требованиям по безопасности информации; возможные нарушения персоналом и др. пользователями ОИ требований нормативных документов по защите; нескоординированность или отсутствие необходимых сил и средств для реализации мер в области защиты информации; возможные ошибки персонала ОИ; нарушения установленных регламентов обработки, передачи и хранения информации о деятельности предприятия; отказы технических и программных средств объекта.

Нарушители установленного регламента применения защищенного ОИ могут быть классифицированы следующим образом:

* реализующие доступные им внутренние угрозы с целью получения НСД к информации и другим защищаемым ресурсам объекта, а также не работающие на ОИ и не допущенные к защищаемой информации;

* реализующие внешние угрозы (в основном, с помощью радиоэлектронных способов съема защищаемой информации);

* лица, привлеченные для оказания различных услуг, но не допущенные к информации и не работающих на ОИ;

* пользователи и обслуживающий персонал объекта, совершающий ошибки в силу нечеткого выполнения служебных обязанностей, приводящих к нарушению уровня защищенности объекта информатизации. Основные категории воздействия: модификация, удаление, задержка, переупорядочивание, дублирование регулярных и посылка ложных сообщений; воспрепятствование передаче сообщений; кража, хищение ценностей; шантаж сотрудников объекта; осуществление ложных соединений и др.

Критерий разбиения на санкционированный и несанкционированный доступ формулируется на этапе создания ОИ. Обычно правила различения санкционированного и несанкционированного доступа следует из политики безопасности, реализованной на конкретном объекте информатизации. Нарушитель, как правило, стремится к скрытности своих действий. Выделяют непосредственный и опосредованный НСД. Угрозы могут осуществляться различными способами :

а) реализация угроз информационными способами:

* противозаконный сбор и использование информации;

* нарушение адресности и своевременности информационного обмена;

* манипулирование информацией (дезинформация, сокрытие или искажение информации);

* использование средств массовой информации с позиций, противоречащих интересам физических, юридических лиц и государства, общественных групп и объединений;

б) реализация угроз техническими способами:

* нарушение технологии обработки информации;

* уничтожение или модификация информации в информационных системах;

* уничтожение или модификация средств обработки информации;

* внедрение или использование аппаратных средств для незаконного прослушивания, просмотра, перехвата информации в технических средствах, линиях связи, помещениях;

* преодоление (обход) мер защиты информации с целью получения охраняемых сведений;

в) реализация угроз физическими способами:

* хищение, разрушение, уничтожение носителей информации с зафиксированной на них информацией;

* физическое подавление сигналов, несущих информацию, в каналах ее передачи и системах обработки;

* хищение, разрушение, уничтожение технологических и аппаратных средств обработки и защиты информации;

* физическое воздействие на средства и системы обработки и защиты информации с целью реализации угроз;

* физическое и моральное воздействие на персонал с целью реализации угроз;

г) реализация угроз организационными способами:

* организация закупок и поставка устаревшей или некачественной информации;

* организация закупок и поставка несовершенных или устаревших ин- формационных технологий и средств обработки информации;

* поставка информационных систем или их элементов с встроенными средствами ("закладками"), вызывающими утечку информации или отказ систем;

* неправомерное ограничение доступа к информации;

* невыполнение требований действующего законодательства и норма- тивных документов в отношении информации;

* невыполнение требований договорных обязательств. Способ реализации угрозы учитывается как квалифицирующее обстоятельство при установлении меры ответственности нарушителя.

Вопросы пресечения угроз, порядок и размер возмещения ущерба при их реализации решаются в административном порядке, а в случаях, превышающих полномочия администрации, - судами.

Воздействия вредоносных программ

Такая программа - осмысленный набор инструкций для какого-либо процессора, может выполнять следующие функции:

* скрывать признаки своего присутствия в программной среде ОИ;

* обладает способностью к самодублированию, ассоциированию себя с другими программами и/или переносу своих фрагментов в иные (не занимаемые изначально указанной программой) области оперативной или внешней памяти;

* обладает способностью разрушать (искажать произвольным образом) код программ (отличных от нее) в оперативной памяти ОИ;

* обладает способностью переносить (сохранять) фрагменты информации из оперативной памяти в некоторых областях оперативной или внешней памяти прямого доступа (локальных или удаленных);

* имеет потенциальную возможность искажать произвольным образом, блокировать и/или подменять выводимой во внешнюю память или в канал связи массив информации, образовавшийся в результате работы прикладных программ или уже находящиеся во внешней памяти массивы данных, либо изменять их параметры.

Вредоносные программы можно условно разделить на:

- компьютерные "вирусы"; - программы типа "программный червь " или "троянский конь" и фрагменты программ типа "логический люк";

- программные закладки или разрушающие программные воздействия (РПВ) - обобщенный класс программ (в смысле отсутствия конкретных признаков) с потенциально опасными последствиями. Кроме того, программные закладки можно классифицировать по методу и месту их внедрения и применения (то есть, по "способу доставки" в систему): - закладки, ассоциированные с программно - аппаратной средой ОИ (основной или расширенные BIOS ПЭВМ); - закладки, ассоциированные с программами первичной загрузки (на-

ходящиеся в Master Boot Record или BOOT - секторах активных разделов) - загрузочные закладки; - закладки, ассоциированные с загрузкой драйверов операционной среды, командного интерпретатора, сетевых драйверов т.е. с загрузкой операционной среды; - закладки, ассоциированные с прикладным программным обеспечением общего назначения (встроенные в клавиатурные и экранные драйверы, программы тестирования ПЭВМ, утилиты и оболочки типа NORTON); - исполняемые модули, содержащие только код закладки (как правило, внедряемые в файлы пакетной обработки типа .BAT); - модули-имитаторы, совпадающие с некоторыми программами, требующими ввода конфиденциальной информации, по внешнему виду) - наиболее характерны для Unix-систем; - закладки, маскируемые под программные средства оптимизационного назначения (архиваторы, ускорители обмена с диском и т.д.); - закладки, маскируемые под программные средства игрового и разв- лекательного назначения (как правило, используются для первичного внедрения закладок).

Программные закладки имеют много общего с классическими вирусами, особенно в части ассоциирования себя с исполняемым кодом (загрузочные вирусы, вирусы-драйверы, файловые вирусы). Кроме того, программные закладки, как и многие известные вирусы классического типа, имеют развитые средства борьбы с отладчиками и дисассемблерами.

Нельзя упускать из вида, что доставка закладок с развлекательным программами в автоматизированную систему организации - путь более чем реальный. Для того, чтобы закладка смогла выполнить какие-либо действия по отношении к прикладной программе или данным, она должна получить управление на себя т.е. процессор должен начать выполнять инструкции (команды), относящиеся к коду закладки.

Это возможно только при одновременном выполнении 2-х условий:

- закладка должна находиться в оперативной памяти до начала работы программы, которая является целью воздействия закладки - следовательно, она должна быть загружена раньше или одновременно с этой программой;

- закладка должна активизироваться по некоторому общему как для закладки, так и для программы событию, т.е. при выполнении ряда условий в программно- аппаратной среде управление должно быть передано на программу-"закладку". Данное событие называется активизирующим. Обычно выполнение указанных условий достигается путем анализа и обработки "закладкой" общих относительно "закладки" и прикладной программы воздействий - как правило, прерываний или сообщений.

Типовой характер опосредованного НСД для информационной системы организации таков: нарушитель производит первичную модификацию "штатной" программно-аппаратной среды платежной системы. В принципе на этом атака может закончиться (так, неоднократно фиксировались попытки статической коррекции программ электронной цифровой подписи (ЭЦП), так чтобы при проверке подписи под документами она всегда оставалась либо верной, либо неверной).

Однако чаще всего на первом этапе нарушитель устанавливает в систему некоторый модуль накопления или анализа циркулирующей в банковской системе информации т.е. фактически перехватывает некоторую информацию.

Нарушитель может накапливать конфиденциальную информацию на двух уровнях: - информация непосредственно клиентов (пароли, (для пластиковых карт ПИН-коды), номера счетов и суммы счетов и т.д.); - персональная информация ответственных исполнителей (ключи ЭЦП, личные пароли, содержание карт или персональных устройств доступа). После получения интересующей информации злоумышленник восстанавливает рабочую среду в исходное состояние, а перехваченную информацию использует сам, либо передает третьим лицам.

Таким образом, осуществление опосредованного НСД (с применение закладок, либо путем коррекции рабочей среды) требует нескольких условий, а именно: физический доступ к аппаратным компонентам банковской системы (в ряде условий возможно удаленное воздействие по сети телекоммуникации), достаточное время, высокая квалификация нарушителя. Из этого следует, что нарушителем может быть сотрудник самой организации,

имеющий доступ к ее программному обеспечению и ПЭВМ, либо сотрудник служб информатизации и телекоммуникации, технического обслуживания и ремонта.

Возможна ситуация, когда нарушитель уже не имеет доступа к базам данных организации, но знает (будучи разработчиком каких-либо подсистем) конкретные его особенности и поддерживает контакт с действующим персоналом. Присутствующая в системе закладка может искажать параметры каких-либо операций, при этом выделяют статическое и динамическое искажение. Статическое искажение заключается в изменении параметров программной среды. Например, редактирование файла AUTOEXEC.BAT так, чтобы первой запускалась заданная злоумышленником программа. Изменение исполняемых модулей (редактирование кода или данных) с целью последующего выполнения нужных злоумышленнику действий. Динамическое искажение заключается в изменении каких-либо параметров процессов при помощи ранее активизированной закладки. Статическое искажение, как правило, проводится один раз. Вообще говоря, разделить эти процессы трудно - так, например, внедрение закладки по вирусному механизму в исполняемый файл сопряжено со статическим искажением программы, а ее дальнейшая работа может быть связана с тем, что код закладки будет резидентным и станет влиять на файловые операции уже динамически.

Угрозы электронно-цифровой подписи

Практика применения ЭЦП в системах автоматизированного документооборота показала, что именно программная реализация ЭЦП наиболее сильно подвержена влиянию со стороны программных закладок, которые позволяют осуществлять проводки заведомо фальшивых документов и вмешиваться в порядок разрешения споров по факту применения ЭЦП.

Отметим четыре основных способа воздействия программных закладок на ЭЦП:

* способ искажения входной информации - связан с искажением поступающего на подпись файла;

* способ искажения результата проверки - связан с влиянием на признак правильности подписи независимо от результата работы;

* способ изменения длины сообщения - предъявление программе ЭЦП электронного документа меньшей длины, следовательно, производится подпись только части документа.

* способ искажения программы ЭЦП - связан с изменением исполняемого кода самой программы ЭЦП (изменение алгоритма хеширования и т.д.). Здесь мы коснулись важной в практическом плане проблеме: безопасности электронного документа, которая во многом базируется на "криптостойкости" алгоритмов электронно-цифровой подписи и протоколов, которые используются при реализации этих алгоритмов.

В последнее время активно развиваются системы электронной торговли. Эти системы также подвержены угрозам информационной безопасности. Рассмотрим некоторые аспекты информационной безопасности систем электронной торговли. Особое значение приобретает угроза информации, связанные с взаимодействием через Интернет или с подключением к всемирной сети.

Здесь следует выявить: - угрозы безопасности по отношению к Интернет-системам; - причины относительно высокой потенциальной уязвимости Инетрнет- систем; - источники угроз Интернет-системам; - основные объекты атак; - комплекс технических средств защиты Интернет-сервисов; - виды атак на финансовые сообщения и финансовые транзакции; - угрозы защищенности в глобальных сетях; - уязвимости криптографических алгоритмов и протоколов, применяемых в Интернет; - атаки хакеров и злоумышленные действия пиратов; - уровень защищенности используемых БД; - возможные уязвимости и недостатки ПО, реализующую электронную коммерцию; - уязвимости сетевых операционных систем; - возможности несанкционированного доступа через сеть; - степень парольной защиты; - возможности уязвимости при управлении ключами; - проблемы при построении межсетевых фильтров; - недостаточная реализация или некорректная разработка политики безопасности; - разрушающие программные воздействия.

Кратко остановимся на некоторых способах защиты от наиболее распространенных в настоящее время угроз. Выделяют следующие способы защиты:

-физические (препятствие)
-законодательные
-управление доступом
-криптографическое закрытие.

Физические способы защиты основаны на создании физических препятствий для злоумышленника, преграждающих ему путь к защищаемой информации (строгая пропускная система на территорию и в помещения с аппаратурой или с носителями информации). Несмотря на богатый опыт по применению таких способов следует признать, что они эффективны только от "внешних" злоумышленников и не защищают информацию от тех лиц, которые обладают правом входа в помещение.

К законодательным средствам защиты относятся законодательные акты, которыми регламентируются правила использования и обработки информации ограниченного доступа и устанавливаются меры ответственности за нарушения этих правил.

Под управлением доступом понимается способ защиты информации регулированием использования всех ресурсов системы (технических, программных, элементов баз данных). В автоматизированных системах информационного обеспечения должны быть регламентированы порядок работы пользователей и персонала, право доступа к отдельным файлам в базах данных и т.д. В сетях ЭВМ наиболее эффективными являются криптографические способы защиты информации.

Если физические способы защиты могут быть преодолены путем, например, дистанционного наблюдения, подключения к сети или подкупа персонала, законодательные не всегда сдерживают злоумышленника, а управление доступом не гарантирует от проникновения изощренных "хакеров", то криптографические методы, если они удовлетворяют соответствующим требованиям, характеризуются наибольшей степенью "прочности". Выбор тех или иных способов защиты информации в автоматизированной системе информационного обеспечения представляет собой сложную оптимизационную задачу, учитывающую вероятность различных угроз информации, стоимость реализации различных способов защиты, наличие различных заинтересованных сторон. В общем случае для нахождения оптимального варианта решения такой задачи необходимо применение теории игр, в частности теории биматричных игр с ненулевой суммой. Для пользователей важно знать, что современная наука имеет методы, позволяющие рекомендовать организации такой набор средств защиты, используя который, можно быть уверенным в том , что при данных затратах максимизируется безопасность информации и наоборот при заданном значении безопасности информации можно выбрать набор средств минимальной стоимости.

Самым распространенным методом установления подлинности является метод паролей. Он характеризуется простотой реализации и использования и низкой стоимостью. Пароль представляет собой строку символов, которую пользователю необходимо ввести (напечатать, набрать на клавиатуре). Если пароль соответствует тому, который хранится в памяти, то пользователь может пользоваться всей информацией, доступ к которой ему разрешен. Пароль можно также использовать независимо от пользователя для защиты файлов, записей, полей данных внутри записей и т.д. Различаются несколько типов паролей: простой пароль, пароль однократного использования, пароль на основе выбора символов, пароль на основе метода "запрос-ответ", пароль на основе определенного алгоритма.

Тема 16. Психологические аспекты взаимодействия с СОИУ

Критерии проектирования:

1) Качество опытов: как эффективный интерактивный дизайн способствует успеху в работе?

2) Понимание пользователей: насколько хорошо группа разработчиков понимает потребности и задачи конечных пользователей? В какой мере данное понимание отражено в программном обеспечении?

3) Эффективность процесса проектирования: является ли продукт действительно обдуманного и тщательного реализованного продукта проектирования?

4) Потребности: что обеспечивает эффективность проекта? Имеет ли продукт общественную, экономическую или другую значимость?

5) Пригодность к изучению и использованию: насколько сложен данный продукт в использовании и обучении? Соответствует ли он своему назначению? Как организована его поддержка? Имеются ли альтернативные пути достижения поставленной цели в зависимости от опыта работы, аналогов и привычек пользователей?

6) Соответствие: соответствует ли дизайн продукта решению поставленных проблем? Отвечает ли продукт требованиям практичности и целесообразности? В какой степени решение проблемы соответствует социальным, культурным, экономическим, техническим аспектам?

7) Эстетическое чувство: насколько пользователю использование продукта? Является ли он цельным с точки зрения дизайна, графики, последовательности действий и информативности? Отвечает ли дизайн технологическим нормам? Удовлетворяет ли он задаче интеграции программного и аппаратного обеспечения?

8) Изменяемость: насколько обоснована способность продукта изменяться? В какой мере продукт соответствует требованиям индивидуального пользователя и группы пользователей? Как проектирование позволяет продукту меняться и подстраиваться под различные, возможно непредвиденные случаи использования?

9) Управляемость: в какой мере продукт помогает пользователям управлять такими процессами, как инсталляция, обучение, сопровождение и т.п.

Принципы проектирования пользовательского интерфейса.

Принципы разработки интерфейса – это высокоуровневые концепции и представления, используемые при проектировании программного обеспечения. В основе принципов разработки лежат физические и ментальные модели пользователей, их психология и психическая возможность.

При проектировании необходимо выделить важнейший принцип, который будет использоваться при поиске компромиссов. Попытка соблюсти все принципы скажется отрицательно на результате.

Можно выделить 3 принципа разработки пользовательского интерфейса:

1) Контроль пользователем интерфейса;

2) Уменьшение загрузки памяти пользователя;

3) Последовательность пользовательского интерфейса.

Правила:

1) Необходимо дать контроль пользователю. Опытные проектировщики позволяют пользователям решать некоторые задачи по-своему собственному решению. Но: необходимо наличие у пользователей необходимых навыков. Если задача решается не пользователем, то он должен иметь возможность контролировать процесс. Принципы, которые дают пользователю контроль над системой: 1) необходимо благоразумно использовать режим. 2) предоставить возможность пользователю выбрать: работать с мышью или клавиатурой, или с тем и другим вместе. 3) необходимо позволить пользователю сфокусировать внимание (прерываемость). 4) полезность: демонстрировать пользователю поясняющие советы и тексты. 5) немедленная обратная связь и обратные действия. 6) необходимо дать возможность пользователю свободно ориентироваться в интерфейсе. 7) интерфейс должен приспосабливаться к пользователям с разными уровнями навыков. 8) пользовательский интерфейс должен быть понятным (прозрачным), т.е. при хорошем интерфейсе пользователь его не воспринимает, а ощущает себя как бы внутри компьютера и может свободно манипулировать объектами.

2) Уменьшить нагрузку на память пользователя. Принципы: 1) не следует кратковременную память. Не следует вынуждать пользователей запоминать и выполнять то, что может сделать и компьютер. 2) необходимо полагаться на распознавание, а не на повторение. Необходимо предусматривать списки и меню, содержащие объекты или документы, которые можно выбрать. Не заставлять пользователей вводить информацию вручную без поддержки системы. 3) необходимо обеспечить визуальные подсказки. Пользователи должны знать, где они находятся, что делают, и что они могут сделать в дальнейшем. Когда пользователи находятся в каком-либо режиме, они должны быть информированы об этом посредствам соответствующих индикаторов. 4) необходимо предусмотреть функцию отмены последнего действия, его повтора и функции установки по умолчанию. Нужно использовать способность компьютера сохранять и отыскивать информацию о выборе пользователя и свойствах системы. Необходимо предусмотреть многоуровневые системы отмены и повтора команд. 5) необходимо реализовывать прямой доступ к элементам интерфейса с помощью клавиатуры. Как только пользователь хорошо освоит программный продукт, он начинает испытывать потребности в ускорителях. Однако, при их реализации нужно следовать стандартам. 6) необходимо использовать синтаксис действий с объектами: объектно-ориентированный синтаксис позволяет пользователю понять взаимосвязь между объектами и действиями в программном продукте. Объектно-ориентированный синтаксис был описан разработчиками Palo Alta Research Center (PARC). Xerex. 7) следует использовать метафоры реального мира. Метафоры позволяют переносить свои знания пользователям из реального мира в мир компьютеров. Если обнаружится, что метафора не отвечает своему назначению во всем интерфейсе, необходимо выбрать новую метафору. Если же метафора выбрана, то следует неукоснительно следовать ей во всем интерфейсе. 8) Нужно объяснять понятия и действия. Пользователи не должны сомневаться по поводу программного перехода. Не нужно показывать все функции, а только те, в которых есть потребность. Необходимо обеспечить легкий доступ к наиболее часто используемым функциям и действиям. Редко используемые функции следует скрыть и позволить пользователю вызывать их по мере необходимости. Для неподготовленных пользователей необходимо использовать режим «Мастер». 9) необходимо увеличить визуальную ясность. необходимо использовать принципы визуального проектирования для облегчения восприятия информации.

3) Последовательность пользовательского интерфейса. Пользователи могут переносить свои знания и навыки из одной программы в другую программу – это преимущества. Принципы создания совместимого интерфейса: 1) проектирование последовательного интерфейса. пользователь должен иметь опорные точки при перемещении в интерфейсе. Это заголовки окон, навигационные карты, древовидная структура. Кроме того, пользователь должен иметь возможность завершить поставленную задачу без изменения среды работы или переключение между стилями ввода данных. 2) общая совместимость всех программ. Совместимость реализуется на трех уровнях: подача информации; поведение программы; техника взаимодействия. Совместимость в подаче информации - пользователь может воспринимать информацию похожую в логическом, визуальном, физическом виде во всей программе. Совместимость в поведении – одни и те же объекты имеют одно и тоже поведение. совместимость в технике взаимодействия – способы работы с мышью и клавиатурой должны быть одинаковы во всех программах. 3) сохранение результатов взаимодействия – при выполнении одних и тех же действий должны получать одни и те же результаты. 4) эстетическая привлекательность и цельность. 5) поощрение изучения.

Тема 17. Управление проектами АСОИУ

Проект – это ограниченное во времени целенаправленное изменение отдельной системы с установленными требованиями к качеству результатов, возможными границами расходов ресурсов и специфической организации. Проект является неповторимым, уникальным

и обладает признаками новизны.

Управление проектом – это планирование, координация и контроль работ по проекту для достижения целей в рамках заданного бюджета, сроков и с надлежащим качеством.

Базовые понятия.

Задача – это общий термин для работы, которая не включена в структуру пооперационного перечня работ, но потенциально может быть разбита на части лицами, ответственными за ее выполнение.

Действие – это элемент работы, выполняемый в процессе реализации проекта. Все действия характеризуются ожидаемой продолжительностью, стоимостью, прогнозируемыми требованиями к ресурсам и может быть разделено на отдельные задачи.

Фаза – группа действий или задач, в ходе осуществления которых производится существенная часть рабочего продукта.

Проект – это уникальное, ориентированное на достижение цели срочное и ограниченное условиями действие.

Программа – совокупность взаимосвязанных проектов.

Система – организованный элемент, выступающий как единое целое.

Управление программными проектами.

Конечным продуктом проектом разработки программного обеспечения является программа. Применяется термин «программный инжениринг» - этот термин от организации SEI (Software Engineering Institute).

Организации, причастные к разработке стандартов в области управления программными проектами и разработки программного обеспечения:

IEEE – Institute of Electrical and Electronics Engineers.

NIST – Nation Institute of Standards and Technology.

ISO – International Organization for Standards.

ANSI – American National Standards Institute.

Инжениринг программного обеспечения.

Это практическое применении научных знаний при разработки и создании компьютерных программ и связанной с ними документации, необходимой для разработки программ, дальнейшего их использования и поддержки.

Определение института IEEE: инжениринг ПО – это систематический подход к развитию, движению, поддержки и прекращению эксплуатации ПО.

При рассмотрении программных проектов оперируют четырьмя переменными:

1) Время;
2) Ресурсы;
3) Объем работ;
4) Качество.

Управлять всеми четырьмя переменными одновременно невозможно.

Задача менеджера проекта – это уравновешивать четыре переменные.

Объем работ в методиках управления другими типами проектов не рассматривается, т.к. задается до начала выполнения проекта и дальше не изменяется.

Характеристики проекта:

Цель проекта – у проекта должно быть четко определена цель или ряд целей, и в результате выполнения проекта должен быть получен результат. Если проект имеет множество целей, то они должны быть связаны и не конфликтовать.

Момент начала и завершения действия, т.е. проект должен иметь четко определенное начало и конец действия. Обычно эта характеристика выражается в виде даты. Поддержка ПО обычно не относится к проекту и им не является.

Процесс управления проектом

Выделяют следующие процессы: 6 основных групп

1. Процессы инициации. Это процессы, связанные с решением о начале выполнения проекта.
2. Процессы планирования. Это определение целей и критериев для оценки результатов проекта и разработка схем их достижения.
3. Процессы исполнения. Это координация людских и других ресурсов для выполнения плана.
4. Прессы анализа. Это определение соответствия плана и исполнения проекта целям и критериям успеха, а также принятие решений о целесообразности управляющих воздействий.
5. Процессы управления. Это определение необходимых корректирующих воздействий, их согласование, утверждение и управление.
6. Процессы завершения. Это формализация выполнения проекта и подведение его к упорядоченному финалу.

Процессы управления связаны своими результатами, т.е. результаты одного процесса становятся исходной информацией для другого.

Кроме того, сам процесс может содержать в себе несколько фаз, в этом случае процесс управления будет относиться к какой-нибудь одной или ко всем фазам одновременно (см. рис).

Взаимосвязи процесса

– Процессы инициации. Единственный процесс, который носит название авторизации, т.е. решение начать следующую фазу проекта.
– Процессы планирования. При планировании выделяют следующие процессы:

1. Планирование цели – это постановка задачи (обоснование цели, этапы и т.д.)
2. Декомпозиция цели. Для разбиения этапов проекта на более мелкие и соответственно более управляемые.
3. Определение состава операции проекта – это составление перечня операций, из которых состоит выполнение различных этапов проекта
4. Определение взаимосвязи операций. Документированное определение связей
5. Оценка длительности и объема работ. Необходимо определить длительность и объем работ каждой из операции
6. Процесс определения ресурсов. Необходимо определить количество ресурсов, которые используются для реализации проекта
7. Назначение ресурсов. Это распределение ресурсов по отдельным операциям проекта
8. Оценка стоимости. Определение того, сколько понадобится средств для реализации каждой операции.
9. Составление расписания работ. Составление графика выполнения работ, определение сроков описания работ, длительности, последовательности, определение временной потребности в ресурсах и т.д.
10. Оценка бюджета. Это получение стоимостных оценок для проекта в целом и для отдельных его фаз.
11. Разработка плана исполнения проекта. Это состав общего интегрального документа, который содержит результаты всех предыдущих процессов.
12. Определение критериев успеха.

Существует ряд дополнительных и вспомогательных процессов планирования:

- ✓ Планирование качества
- ✓ Планирование организации
- ✓ Назначение персонала
- ✓ Планирование взаимодействий
- ✓ Идентификация риска
- ✓ Разработка реагирования (определение необходимых действий для предупреждения риска и реагирования окружающих)
- ✓ Планирование поставок
- ✓ Подготовка условий (требования к поставкам и определение потенциальных поставщиков)

– Процессы исполнения. Под исполнением понимаются процессы, связанные с реализацией плана проекта. Исполнение должно изменяться для выявления отклонения от плана.

Контроль исполнения также относится к группе процессов исполнения. К основным процессам относится сам процесс исполнения плана проекта.

Вспомогательные процессы:

- ✓ Учет исполнений (подготовка необходимых для участников проекта информации с требуемой периодичностью)
- ✓ Подтверждение качества. Это регулярная проверка исполнения проекта с целью подтверждения соответствия принятому стандарту качества.
- ✓ Выбор поставщиков (выбор поставщиков, их анализ, заключение контракта)
- ✓ Контроль контракта. Это контроль исполнения контракта поставщиками и подрядчиками.
- ✓ Подготовка предложений. Сбор заявок, рекомендаций и т.д., направленных на улучшение хода реализации
- ✓ Развитие команды проекта (повышение квалификации участников команды)

– Процессы анализа. Они включают в себя как анализ плана, так и анализ исполнения.

Анализ плана означает определение того, соответствует ли план предъявляемым проекту требованиям и ожиданиям участников проекта.

Анализ исполнения предназначен для оценки состояния и прогноза успешности исполнения проекта.

Основные процессы анализа:

1. Анализ сроков
2. Анализ стоимости
3. Анализ качества
4. Анализ подтверждения целей

Вспомогательные процессы:

- ✓ Оценка исполнения – это анализ результатов работы и распределения проектной информации с целью информирования участников проекта от использования ресурсов для достижения целей проекта
- ✓ Анализ ресурсов – это оценка фактической и прогнозной фиктивности использования ресурсов.

– Процессы управления. Это определение и применение необходимых управляющих воздействий с целью успешной реализации проекта.

Основные:

1. общее управление изменениями
2. управление ресурсами
3. управление целями

4. Управление качеством

Вспомогательные процессы:

– Управление рисками
– Управление контрактами

– Процессы завершения

1. Завершение (закрытие) контрактов
2. Административное завершение – это подготовка, сбор и распределение информации, необходимой для формального завершения проекта.