Федеральное агентство по образованию РФ
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ГОУВПО «АмГУ»)

УТВЕРЖДАЮ
Зав. кафедрой ИУС
_____А.В. Бушманов
«_____»_____

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС

по дисциплине «Элементы моделирования и проектирования информационных систем»
для студентов специальности 230201 – Информационные системы и технологии

Составитель ассистент кафедры ИУС Жилиндина О.В.

Факультет математики и информатики

Кафедра информационных и управляющих систем

2007

Печатается по решению
редакционно-издательского совета
факультета математики и информатики
Амурского государственного
университета

О.В. Жилиндина

Учебно-методический комплекс по дисциплине «Элементы моделирования и проектирования информационных систем». Для студентов специальности 230201 «Информационные системы и технологии» очной формы обучения.- Благовещенск: Амурский гос. ун-т, 2007.

Пособие содержит рабочую программу, курс лекций, методические рекомендации по проведению и выполнению лабораторных работ. Составлено в соответствии с требования государственного образовательного стандарта.

I. РАБОЧАЯ ПРОГРАММА

курс 3 семестр 6

Лекции 36 (час.) Экзамен 6 семестр

Лабораторные занятия 18 (час.)

Самостоятельная работа 56 (час.)

Всего часов 110 час.

1. ЦЕЛИ И ЗАДАЧИ ДИСЦИПЛИНЫ, ЕЕ МЕСТО В УЧЕБНОМ ПРОЦЕССЕ

Программное обеспечение, как и люди, имеет жизненный цикл. Под полным жизненным циклом программного обеспечения обычно понимают процесс его развития, состоящий из четырех последовательных фаз, которые называются "начало", "исследование", "проектирование" и "внедрение". В свою очередь фазы делятся на этапы. Разработка программного обеспечения - это процесс, охватывающий три первые фазы его жизненного цикла. В ходе разработки постепенно создается программный продукт, готовый к внедрению. Разработка завершается тестированием программного комплекса и принятием решения о готовности программ к внедрению.

Сегодня программное обеспечение разрабатывается с помощью типовых технологий. Они называются методологиями программирования или просто методами программирования. С точки зрения задач, решаемых в нашем Курсе,

наибольший интерес представляет метод объектно-ориентированного программирования (ООП), по-английски object-oriented programming (OOP). С его помощью создаются программы, написанные на объектно-ориентированных языках программирования, таких как C++, Java и др.

Программное обеспечение - это важная, но не единственная часть проектируемой компьютерной информационной системы. Другой ее частью является предметная область. Поэтому для разработки информационных систем, наряду с методом ООП, используются также метод объектно-ориентированного проектирования (object-oriented design, OOD) и метод объектно-ориентированного анализа (object-oriented analysis, OOA). С помощью методов OOA и OOD моделируются предметные области информационных систем, которые у нас в России, иногда, называют объектами автоматизации.

Первая часть курса лекций посвящена визуальным (наглядным) и математическим моделям методов ООП, OOA и OOD. Визуальные модели это такие модели, которые можно рисовать на бумаге или экранах компьютеров. Математические модели это формальные модели, представляемые в виде символьных соотношений.

По определению одного из создателей языка UML (Unified Modelling Language) Гради Буча:

Объектно-ориентированное программирование (ООП) - это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Объектно-ориентированное проектирование (OOD) - это методология проектирования, соединяющая в себе процесс объектной декомпозиции и приемы представления логической и физической, а также статической и динамической моделей проектируемой системы.

Объектно-ориентированный анализ (ООА) – это методология, при использовании которой требования к проектируемой системе воспринимаются с точки зрения классов и объектов, выявленных в предметной области.

Основные понятия: программное обеспечение, методология программирования, объектно-ориентированное проектирование, объектно-ориентированное программирование, язык UML, модель IDEF0, модель IDEF3, модель DFD, логический уровень модели данных, физический уровень модели данных.

2. СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

2.1. Федеральный компонент

Программа курса «Элементы моделирования и проектирования информационных систем» составлена в соответствие с требованиями государственного образовательного стандарта специализации – Информационные системы и технологии, специализации 230201, блок дисциплин специализации ДС 03.

2.2. Содержание лекций – 36 часов

1. Создание моделей бизнесс-процессов. – 2 часа.
2. Диаграммы IDEF0. – 2 часа.
3. Диаграммы потоков данных. – 2 часа
4. Диаграммы последовательности действий. – 2 часа.
5. Методология процедурно-ориентированного программирования. – 2 часа
6. Исторический обзор развития методологии объектно-ориентированного анализа и проектирования сложных систем. – 2 часа.
7. Основные компоненты языка UML. – 2 часа.
8. .Диаграмма вариантов использования – 2 часа
9. Диаграмма классов. – 4 часа
10. Диаграмма состояний. – 4 часа
11. Диаграмма деятельности. – 4 часа.

12. Диаграмма последовательности. – 4 часа.
13. Диаграмма кооперации. – 4 часа.
14.

2.3. Лабораторные работы – 18 часов

1. Теоретическое введение в предметную область. – 2 часа.
2. Методология IDEF0. – 2 часа.
3. Дополнение моделей процессов диаграммами IDEF3 и DFD. – 2 часа.
4. Создание отчетов. – 2 часа.
5. Создание диаграммы вариантов использования – 2 часа.
6. Создание диаграммы Классов. – 2 часа.
7. Создание диаграмм Взаимодействия. – 2 часа.
8. Создание диаграммы состояния. – 2 часа.
9. Создание диаграмм пакетов, компонентов и размещения. – 2 часа.

2.4. Вопросы к экзамену

1. Принципы построения модели IDEF0
2. Диаграммы IDEF0
3. Диаграмма потоков данных
4. Метод описания процессов IDEF3
5. Перекрестки
6. Язык UML и его назначение
7. Диаграмма вариантов использования
8. Вариант использования
9. Актер
10. Интерфейс
11. Отношения на диаграмме вариантов использования
12. Диаграмма классов

13. Класс
14. Отношения между классами
15. Диаграмма состояний
16. Автоматы
17. Состояния
18. Переход, сложные переходы
19. Диаграмма деятельности
20. Состояния действия
21. Переходы
22. Дорожки
23. Объекты

2.5. Темы для самостоятельной работы

1. Моделирование и проектирование системы «Маркетинговые исследования в корпорации».
2. Моделирование и проектирование системы «Управление арендой автотранспортных средств».
3. Моделирование и проектирование системы «Продажа билетов пассажирам авиационного транспорта».
4. Моделирование и проектирование системы «Система управления кораблем».

ВИДЫ КОНТРОЛЯ

Текущий контроль осуществляется во время проведения лабораторных работ посредством приема отчетов, устного опроса, коллоквиумов. Промежуточный контроль осуществляется путем проведения коллоквиумов. Итоговый

контроль проводится после успешного текущих и промежуточных контролей в виде устного или письменного экзамена.

КРИТЕРИИ ОЦЕНОК ЗНАНИЙ СТУДЕНТОВ

Отлично

Студент дает полные ответы на теоретические вопросы билета, показывая глубокое знание учебного материала, свободное владение основными понятиями и терминологией; ответ на дополнительный вопрос.

Хорошо

Студент дает ответы на теоретические вопросы билета, показывая прочное знание учебного материала, владение основными понятиями и терминологией; ответ на дополнительный вопрос.

Удовлетворительно

Студент дает неполные ответы на теоретические вопросы билета, показывая поверхностное знание учебного материала, владение основными понятиями и терминологией; при неверном ответе на билет ответы на наводящие вопрос.

Неудовлетворительно

Студент не дает полные ответы на теоретические вопросы билета, показывая лишь фрагментарное знание учебного материала, незнание основных понятий и терминологии; наводящие вопросы остаются без ответа.

3. УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ

Основная литература

1. А.М. Вендров. Проектирование программного обеспечения экономических информационных систем М.: Финансы и статистика, 2001.

2. С.В. Черемных, И.О. Семенов, В.С. Ручкин. Моделирование и анализ систем. IDEF – технологии: практикум. М.: Финансы и статистика, 2002.

Дополнительная литература

1. Грейди Буч, Джеймс Рамбо, Айвар Джекобсон. Язык UML. Руководство пользователя. Питер, 2004.
2. Джозеф Шмуллер. Освой самостоятельно UML за 24 часа. Вильямс, 2002.
3. Дуг Розенберг, Кендалл Скот. Применение объектного моделирования с использованием UML и анализ прецедентов, ДМК Пресс, 2002.
4. Кендал Скотт. UML основные концепции. Вильямс, 2002.
5. Лешек А. Мацяшек. Анализ требований и проектирование систем. Разработка информационных систем с использованием UML. Вильямс, 2002.
6. Мартин Фаулер, Кендалл Скотт. UML. Основы. Краткое руководство по унифицированному языку моделирования. Символ-плюс, 2002.
7. М. Фаулер, К. Скотт. UML в кратком изложении. Применение стандартного языка объектного моделирования. Мир, 1999.
8. Терри Кватрани. Визуальное моделирование с помощью Rational Rose 2002 и UML.Вильямс, 2003.
9. Уэнди Боггс, Майкл Боггс. UML и Rational Rose. Лори, 2000.
10. Хассан Гома. UML. Проектирование систем реального времени, распределенных и параллельных приложений. ДМК, 2002.

4. ТЕХНИЧЕСКИЕ И ПРОГРАММНЫЕ СРЕДСТВА ОБЕСПЕЧЕНИЯ ДИСЦИПЛИНЫ:

Лекции проводятся в стандартной аудитории, оснащенной в соответствии с требованиями преподавания теоретических дисциплин.

Для проведения лабораторных работ необходим компьютерный класс на 12-14 посадочных рабочих мест пользователей. В классе должны быть установлены программы BPWin, ERWin, Rational Rose.

5. УЧЕБНО-МЕТОДИЧЕСКАЯ (ТЕХНОЛОГИЧЕСКАЯ) КАРТА ДИСЦИПЛИНЫ

Номер недели	Вопросы, изучаемые на лекции	Занятия (номера)		Используемые нагляд. и метод. пособия	Самостоятельная работа студентов		Форма контроля
		Практич (семин.)	Лабо-рат.		Содержание	часы	
1	1		1		Создание диаграммы IDEF0	3	отчет
2	2		1			3	отчет
3	3		2			3	отчет
4	3		2			4	отчет
5	4		3		Создание диаграммы DFD	3	отчет
6	5		3			3	отчет
7	6		4			3	отчет
8	6		4		Создание диаграммы IDEF3	3	отчет
9	7		5			4	отчет
10	7		5			3	отчет
11	8		6		Создание модели в ERWin	3	отчет
12	9		6			4	отчет
13	10		7			3	отчет
14	10		7			4	отчет
15	11		8			4	отчет
16	12		8		Создание отчетов	3	отчет
17	12		9			4	отчет
18	13		9			3	отчет

II. ГРАФИК ВЫПОЛНЕНИЯ И МЕТОДИЧЕСКИЕ УКАЗАНИЯ К САМОСТОЯТЕЛЬНОЙ РАБОТЕ СТУДЕНТОВ

Для самостоятельной работы студентам предлагается выбрать одну из предметных областей (список находится в рабочей программе). Возможно и предложение студентом своей предметной области.

Студент должен изучить выбранную область, рассмотреть функциональную модель, документооборот в ней. И оформить в виде диаграмм.

III. КРАТКИЙ КОНСПЕКТ ЛЕКЦИЙ

Лекция 1 (2 часа)

СОЗДАНИЕ МОДЕЛЕЙ БИЗНЕС-ПРОЦЕССОВ

Инструментальная среда BPWin.

BPWin поддерживает три методологии – IDEF0, IDEF3, DFD, каждая из которых решает свои специфические задачи. В BPWin возможно построение смешанных моделей, т.е. модель может содержать одновременно как диаграммы IDEF0 , так и диаграммы IDEF3 DFD. Модель BPWin рассматривается как совокупность работ, каждая из которых оперирует некоторым набором данных.

Инструмент навигации Model Explorer (который находится в левой стороне основного окна BPWin) имеет три вкладки – Activities, Diagrams, Objects.

Принципы построения модели IDEF0

На начальных этапах создания ИС необходимо понять, как работает организация, которую собираются автоматизировать. Для описания работы предприятия необходимо построить модель. Такая модель должна быть адекватна предметной области; следовательно, она должна содержать в себе знания всех участников бизнес-процессов организации.

Процесс моделирования какой-либо системы в IDEF0 начинается с определения контекста, т.е. наиболее абстрактного уровня описания системы в целом. В контекст входит определение субъекта моделирования, цели и точки зрения на модель.

Модель не может быть построена без четко сформулированной цели. Модель должна строиться с единой точки зрения. Целью построения функциональных моделей обычно является выявление наиболее слабых и уязвимых мест деятельности организации, анализ преимуществ новых бизнес-процессов и степени изменения существующей структуры организации бизнеса.

Лекция 2 (2 часа)

ДИАГРАММЫ IDEF0.

Основу методологии IDEF0 составляет графический язык описания бизнес-процессов. Модель в нотации IDEF0 представляет собой совокупность иерархически упорядоченных и взаимосвязанных диаграмм. Каждая диаграмма является единицей описания системы и располагается на отдельном листе.

Модель может содержать четыре типа диаграмм:

- контекстную (в каждой модели может быть только одна контекстная диаграмма);

- декомпозиции;

- дерева узлов;

- только для экспозиции (FEO).

Контекстная диаграмма является вершиной древовидной структуры диаграмм и представляет собой самое общее описание системы и ее взаимодействия с внешней средой. После описания системы в целом проводится разбиение ее на крупные фрагменты. Этот процесс называется функциональной декомпозицией, а диаграммы, которые описывают каждый фрагмент и взаимодействие фрагментов, называются диаграммами декомпозиции. После декомпозиции контекстной диаграммы проводится декомпозиция каждого большого фрагмента системы на более мелкие и т.д. до достижения нужного уровня подробности описания.

Диаграмма дерева узлов показывает иерархическую зависимость работ, но не взаимосвязи между работами. Диаграмм деревьев узлов может быть в модели сколь угодно много, поскольку дерево может быть построено на произвольную глубину и не обязательно с корня.

Диаграммы для экспозиции (FEO) строятся для иллюстрации отдельных фрагментов модели, для иллюстрации альтернативной точки зрения либо для специальных целей.

Лекция 3 (4 часа)

ДОПОЛНЕНИЕ СОЗДАННОЙ МОДЕЛИ ПРОЦЕССОВ ОРГАНИЗАЦИОННЫМИ ДИАГРАММАМИ, ДИАГРАММАМИ DFD И IDEF3

Диаграммы потоков данных

Диаграммы потоков данных (Data Flow Diagramming, DFD) используются для описания документооборота и обработки информации. Их можно использовать как дополнение к модели IDEF0 для более наглядного отображения текущих операций документооборота в корпоративных системах обработки информации. DFD описывает:

- функции обработки информации (работы);

- документы (стрелки), объекты, сотрудников или отделы, которые участвуют в обработки информации;

- внешние ссылки, которые обеспечивают интерфейс с внешними объектами, находящимися за границами моделируемой системы;

- таблицы для хранения документов.

Метод описания процессов IDEF3

Для описания логики взаимодействия информационных потоков более подходит IDEF3, называемая также workflow diagramming – методологией моделирования, использующая графическое описание информационных потоков, взаимоотношений между процессами обработки информации и объектов, являющихся частью этих процессов. Диаграммы Workflow могут быть использованы в моделировании бизнес – процессов для анализа завершенности процедур обработки информации. С их помощью можно описывать сценарии действий сотрудников организации, например последовательность обработки заказа или события, которые необходимо обработать за конечное время. Каждый сценарий сопровождается описанием процесса и может быть использован для документирования каждой функции.

IDEF3 – это метод имеющий основной целью дать возможность аналитикам описать ситуацию, когда процессы выполняются в определенной последовательности, а также описать объекты, участвующие совместно в одном процессе.

Лекция 4 (2 часа)

МЕТОДОЛОГИЯ ПРОЦЕДУРНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

Основой данной методологии разработки программ являлась процедурная или алгоритмическая организация структуры программного кода. Исходным понятием этой методологии являлось понятие алгоритма, под которым, в общем случае, понимается некоторое предписание выполнить точно определенную последовательность действий, направленных на достижение заданной цели или решение поставленной задачи.

С этой точки зрения вся история математики тесно связана с разработкой тех или иных алгоритмов решения актуальных для своей эпохи задач.

Однако потребности практики не всегда требовали установления вычислимости конкретных функций или разрешимости отдельных задач. В языках программирования возникло и закрепилось новое понятие процедуры.

Методология объектно-ориентированного программирования

Трудоемкость разработки программных приложений на начальных этапах программирования оценивалась значительно ниже реально затрачиваемых усилий, что служило причиной дополнительных расходов и затягивания окончательных сроков готовности программ. В процессе разработки приложений изменялись функциональные требования заказчика, что еще более отдаляло момент окончания работы программистов. Увеличение размеров программ приводило к необходимости привлечения большего числа программистов, что, в свою очередь, потребовало дополнительных ресурсов для организации их согласованной работы.

Во второй половине 80-х годов возникла настоятельная потребность в новой методологии программирования, которая была бы способна решить весь этот комплекс проблем. Такой методологией стало объектно-ориентированное программирование (ООП).

Фундаментальными понятиями ООП являются понятия класса и объекта. При этом под классом понимают некоторую абстракцию совокупности объек-

тов, которые имеют общий набор свойств и обладают одинаковым поведением. Каждый объект в этом случае рассматривается как экземпляр соответствующего класса. Объекты, которые не имеют полностью одинаковых свойств или не обладают одинаковым поведением, по определению, не могут быть отнесены к одному классу.

Основными принципами ООП являются наследование, инкапсуляция и полиморфизм.

Широкое распространение методологии ООП оказало влияние на процесс разработки программ. Наиболее существенным обстоятельством в развитии методологии ООП явилось осознание того факта, что процесс написания программного кода может быть отделен от процесса проектирования структуры программы.

Методология объектно-ориентированного анализа и проектирования

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. 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.

Для выделения или идентификации компонентов предметной области было предложено несколько способов и правил. (

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 новый тип специалиста, который получил название аналитика или архитектора.

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.

Методология ООАП тесно связана с концепцией автоматизированной разработки программного обеспечения (Computer Aided Software Engineering, CASE).

Методология 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.

Важнейшими характеристиками любой системы являются ее структура и процесс функционирования. Процесс функционирования системы тесно связан с изменением ее свойств или поведения во времени.

Процесс функционирования системы отражает поведение системы во времени и может быть представлен как последовательное изменение ее состояний:

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.

Лекция 5 (2 часа)

ИСТОРИЧЕСКИЙ ОБЗОР РАЗВИТИЯ МЕТОДОЛОГИИ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО АНАЛИЗА И ПРОЕКТИРОВАНИЯ СЛОЖНЫХ СИСТЕМ

Предыстория. Математические основы

Представление различных понятий окружающего нас мира при помощи графической символики уходит своими истоками в глубокую древность.

Построение моделей сложных систем, отражающих десятки различных типов объектов и связей между ними, привело в конце 80-х годов к появлению большого числа различных графических нотаций, которые в той или иной степени были ориентированы на решение специальных классов задач. Наибольшее распространение в эти годы получил подход к моделированию программных систем, который назвали системным структурным анализом (ССА). Поскольку многие идеи ССА оказали непосредственное влияние на развитие языка UML, а используемая графическая нотация была реализована в некоторых CASE-средствах, ниже приводится краткая характеристика основных компонентов данного направления графического моделирования программных систем.

Диаграммы структурного системного анализа

Под структурным системным анализом принято понимать метод исследования системы, который начинается с наиболее общего ее описания с последующей детализацией представления отдельных аспектов ее поведения и функционирования. При этом общая модель системы строится в виде некоторой иерархической структуры, которая отражает различные уровни абстракции с ограниченным числом компонентов на каждом из уровней. Одним из главных принципов структурного системного анализа является выделение на каждом из уровней абстракции только наиболее существенных компонентов или элементов системы.

В рамках данного направления программной инженерии принято рассматривать три графические нотации, получивших названия диаграмм: диаграммы "сущность-связь" (Entity-Relationship Diagrams, ERD), диаграммы функционального моделирования (Structured Analysis and Design Technique, SADT) и диаграммы потоков данных (Data Flow Diagrams, DFD).

Модель IDEF-SADT представляет собой серию иерархически взаимосвязанных диаграмм с сопроводительной документацией, которая разбивает исходное представление сложной системы на отдельные составные части.

В настоящее время диаграммы структурного системного анализа IDEF-SADT продолжают использоваться целым рядом организаций для построения и детального анализа функциональной модели существующих на предприятии бизнес-процессов, а также для разработки новых бизнес-процессов.

Модель системы в контексте DFD представляется в виде некоторой информационной модели, основными компонентами которой являются различные потоки данных, которые переносят информацию от одной подсистемы к другой. Каждая из подсистем выполняет определенные преобразования входного потока данных и передает результаты обработки информации в виде потоков данных для других подсистем.

Информационная модель системы в нотации DFD строится в виде диаграмм потоков данных, которые графически представляются с использованием соответствующей системы обозначений.

Основные этапы развития UML

Отдельные языки объектно-ориентированного моделирования стали появляться в период между серединой 1970-х и концом 1980-х годов, когда различные исследователи и программисты предлагали свои подходы к ООАП. В период между 1989-1994 гг. общее число наиболее известных языков моделирования возросло с 10 до более чем 50. Многие пользователи испытывали серьезные затруднения при выборе языка ООАП, поскольку ни один из них не удовлетворял всем требованиям, предъявляемым к построению моделей сложных систем.

К середине 1990-х некоторые из методов были существенно улучшены и приобрели самостоятельное значение при решении различных задач ООАП. Наиболее известными в этот период становятся:

- Метод Гради Буча (Grady Booch), получивший условное название Booch или Booch'91, Booch Lite (позже - Booch'93).

- Метод Джеймса Румбаха (James Rumbaugh), получивший название Object Modeling Technique - OMT (позже - OMT-2).
- Метод Айвара Джекобсона (Ivar Jacobson), получивший название Object-Oriented Software Engineering - OOSE.

Каждый из этих методов был ориентирован на поддержку отдельных этапов ООАП.

История развития языка UML берет начало с октября 1994 года.

Инструментальные CASE-средства и диапазон их практического применения в большой степени зависят от удачного определения семантики и нотации соответствующего языка моделирования. Специфика языка UML заключается в том, что он определяет семантическую метамодель, а не модель конкретного интерфейса и способы представления или реализации компонентов.

Из более чем 800 компаний и организаций, входящих в настоящее время в состав консорциума OMG, особую роль продолжает играть Rational Software Corporation, которая стояла у истоков разработки языка UML. Эта компания разработала и выпустила в продажу одно из первых инструментальных CASE-средств Rational Rose 98, в котором был реализован язык UML.

Язык UML ориентирован для применения в качестве языка моделирования различными пользователями и научными сообществами для решения широкого класса задач ООАП.

На основе технологии UML Microsoft, Rational Software и другие поставщики средств разработки программных систем разработали единую информационную модель, которая получила название UML Information Model.

В настоящее время разработаны средства визуального программирования на основе UML, обеспечивающие интеграцию, включая прямую и обратную генерацию кода программ, с наиболее распространенными языками и средами программирования, такими как MS Visual C++, Java, Object Pascal/Delphi, Power Builder, MS Visual Basic, Forte, Ada, Smalltalk.

Имеются все основания предполагать, что в ближайшие годы язык UML в его современном виде станет основой для разработки и реализации во многих

перспективных инструментальных средствах: в RAD-средствах визуального и имитационного моделирования, а также в CASE-средствах самого различного целевого назначения. Более того, заложенные в языке UML потенциальные возможности могут быть использованы не только для объектно-ориентированного моделирования систем, но и для представления знаний в интеллектуальных системах, которыми, по существу, станут перспективные сложные программно-технологические комплексы.

Лекция 6 (2 часа)

ОСНОВНЫЕ КОМПОНЕНТЫ ЯЗЫКА UML

Язык UML представляет собой общецелевой язык визуального моделирования, который разработан для спецификации, визуализации, проектирования и документирования компонентов программного обеспечения, бизнес-процессов и других систем.

Язык UML основан на некотором числе базовых понятий.

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.

Другим принципом построения моделей сложных систем является принцип многомодельности.

Еще одним принципом прикладного системного анализа является принцип иерархического построения моделей сложных систем.

Таким образом, процесс ООАП можно представить как поуровневый спуск от наиболее общих моделей и представлений концептуального уровня к более частным и детальным представлениям логического и физического уровня. При этом на каждом из этапов ООАП данные модели последовательно до-

полняются все большим количеством деталей, что позволяет им более адекватно отражать различные аспекты конкретной реализации сложной системы.

Назначение языка UML

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.

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

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

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

5. Поощрять развитие рынка объектных инструментальных средств.

6. Способствовать распространению объектных технологий и соответствующих понятий ООАП.

7. Интегрировать в себя новейшие и наилучшие достижения практики ООАП.

Общая структура языка UML

С самой общей точки зрения описание языка UML состоит из двух взаимодействующих частей, таких как:

- Семантика языка UML. Представляет собой некоторую метамодель, которая определяет абстрактный синтаксис и семантику понятий объектного моделирования на языке UML.
- Нотация языка UML. Представляет собой графическую нотацию для визуального представления семантики языка UML.

Семантика определяется для двух видов объектных моделей: структурных моделей и моделей поведения.

Для решения столь широкого диапазона задач моделирования разработана достаточно полная семантика для всех компонентов графической нотации. Требования семантики языка UML конкретизируются при построении отдельных видов диаграмм, последовательное рассмотрение которых служит темой второй части книги. Нотация языка UML включает в себя описание отдельных семантических элементов, которые могут применяться при построении диаграмм.

Формальное описание самого языка UML основывается на некоторой общей иерархической структуре модельных представлений, состоящей из четырех уровней:

- Мета-метамодель
- Метамодель
- Модель
- Объекты пользователя

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 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)

Из перечисленных выше диаграмм некоторые служат для обозначения двух и более других подвидов диаграмм. При этом в качестве самостоятельных представлений в языке UML используются следующие диаграммы:

1. Диаграмма вариантов использования.
2. Диаграмма классов.
3. Диаграмма состояний.
4. Диаграмма деятельности.
5. Диаграмма последовательности.
6. Диаграмма кооперации.
7. Диаграмма компонентов.
8. Диаграмма развертывания.

Большинство перечисленных выше диаграмм являются в своей основе графами специального вида, состоящими из вершин в форме геометрических фигур, которые связаны между собой ребрами или дугами. Поскольку информация, которую содержит в себе граф, имеет в основном топологический характер, ни геометрические размеры, ни расположение элементов диаграмм (за некоторыми исключениями, такими как диаграмма последовательностей с метрической осью времени) не имеют принципиального значения.

Лекция 7 (2 часа)

ДИАГРАММА ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ (USE CASE DIAGRAM)

Визуальное моделирование в UML можно представить как некоторый процесс поуровневого спуска от наиболее обшей и абстрактной концептуальной модели исходной системы к логической, а затем и к физической модели соответствующей программной системы. Для достижения этих целей вначале строится модель в форме так называемой диаграммы вариантов использования (use case diagram), которая описывает функциональное назначение системы или, другими словами, то, что система будет делать в процессе своего функционирования. Диаграмма вариантов использования является исходным концептуальным представлением или концептуальной моделью системы в процессе ее проектирования и разработки.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью так называемых вариантов использования. При этом актером (actor) или действующим лицом называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик. В свою очередь, вариант использования (use case) служит для описания сервисов, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой.

В самом общем случае, диаграмма вариантов использования представляет собой граф специального вида, который является графической нотацией для представления конкретных вариантов использования, актеров, возможно некоторых интерфейсов, и отношений между этими элементами. При этом отдельные компоненты диаграммы могут быть заключены в прямоугольник, который обозначает проектируемую систему в целом. Следует отметить, что отношениями данного графа могут быть только некоторые фиксированные типы взаимосвязей между актерами и вариантами использования, которые в совокупно-

сти описывают сервисы или функциональные требования к моделируемой системе.

Вариант использования

Конструкция или стандартный элемент языка UML вариант использования применяется для спецификации общих особенностей поведения системы или любой другой сущности предметной области без рассмотрения внутренней структуры этой сущности. Каждый вариант использования определяет последовательность действий, которые должны быть выполнены проектируемой системой при взаимодействии ее с соответствующим актером. Диаграмма вариантов может дополняться пояснительным текстом, который раскрывает смысл или семантику составляющих ее компонентов. Такой пояснительный текст получил название примечания или сценария.

Цель варианта использования заключается в том, чтобы определить законченный аспект или фрагмент поведения некоторой сущности без раскрытия внутренней структуры этой сущности. В качестве такой сущности может выступать исходная система или любой другой элемент модели, который обладает собственным поведением, подобно подсистеме или классу в модели системы.

Варианты использования описывают не только взаимодействия между пользователями и сущностью, но также реакции сущности на получение отдельных сообщений от пользователей и восприятие этих сообщений за пределами сущности.

Актеры

Актер представляет собой любую внешнюю по отношению к моделируемой системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или решения частных задач. При этом актеры служат для обозначения согласованного множества ролей, которые могут играть пользователи в процессе взаимодействия с проектируемой системой. Каждый актер может рассматриваться как некая отдельная роль относительно конкретного варианта использования.

Любая сущность, которая согласуется с подобным неформальным определением актера, представляет собой экземпляр или пример актера. Для моделируемой системы актерами могут быть как субъекты-пользователи, так и другие системы. Поскольку пользователи системы всегда являются внешними по отношению к этой системе, то они всегда представляются в виде актеров.

Так как в общем случае актер всегда находится вне системы, его внутренняя структура никак не определяется.

Интерфейсы

Интерфейс (interface) служит для спецификации параметров модели, которые видимы извне без указания их внутренней структуры. В языке UML интерфейс является классификатором и характеризует только ограниченную часть поведения моделируемой сущности. Применительно к диаграммам вариантов использования, интерфейсы определяют совокупность операций, которые обеспечивают необходимый набор сервисов или функциональности для актеров. Интерфейсы не могут содержать ни атрибутов, ни состояний, ни направленных ассоциаций. Они содержат только операции без указания особенностей их реализации. Формально интерфейс эквивалентен абстрактному классу без атрибутов и методов с наличием только абстрактных операций.

Важность интерфейсов заключается в том, что они определяют стыковочные узлы в проектируемой системе, что совершенно необходимо для организации коллективной работы над проектом.

Примечания

Примечания (notes) в языке UML предназначены для включения в модель произвольной текстовой информации, имеющей непосредственное отношение к контексту разрабатываемого проекта. В качестве такой информации могут быть комментарии разработчика (например, дата и версия разработки диаграммы или ее отдельных компонентов), ограничения (например, на значения отдельных связей или экземпляры сущностей) и помеченные значения. Применительно к диаграммам вариантов использования примечание может носить самую общую информацию, относящуюся к общему контексту системы.

Отношения на диаграмме вариантов использования

Между компонентами диаграммы вариантов использования могут существовать различные отношения, которые описывают взаимодействие экземпляров одних актеров и вариантов использования с экземплярами других актеров и вариантов. Один актер может взаимодействовать с несколькими вариантами использования. В свою очередь один вариант использования может взаимодействовать с несколькими актерами, предоставляя для всех них свой сервис.

В языке UML имеется несколько стандартных видов отношений между актерами и вариантами использования:

- Отношение ассоциации (association relationship)
- Отношение расширения (extend relationship)
- Отношение обобщения (generalization relationship)
- Отношение включения (include relationship)

При этом общие свойства вариантов использования могут быть представлены тремя различными способами, а именно с помощью отношений расширения, обобщения и включения.

Лекция 8 (4 часа)

ДИАГРАММА КЛАССОВ (CLASS DIAGRAM)

Центральное место в 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 может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений. На данной диаграмме не указывается информация о временных аспектах функционирования системы. С этой точки зрения диаграмма классов является дальнейшим развитием концептуальной модели проектируемой системы.

Диаграмма классов представляет собой некоторый граф, вершинами которого являются элементы типа "классификатор", которые связаны различными типами структурных отношений.

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 их составляющими компонентами. При этом отдельные компоненты этой диаграммы могут образовывать пакеты для представления более общей модели системы. Если диаграмма классов является частью некоторого пакета, то ее компоненты должны соответствовать элементам этого пакета, включая возможные ссылки на элементы из других пакетов.

Класс

Класс (class) в языке UML служит для обозначения множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов.

Отношения между классами

Кроме внутреннего устройства или структуры классов на соответствующей диаграмме указываются различные отношения между классами. При этом совокупность типов таких отношений фиксирована в языке UML и предопределена семантикой этих типов отношений. Базовыми отношениями или связями в языке UML являются:

- Отношение зависимости (dependency relationship)
- Отношение ассоциации (association relationship)
- Отношение обобщения (generalization relationship)
- Отношение реализации (realization relationship)

Каждое из этих отношений имеет собственное графическое представление на диаграмме, которое отражает взаимосвязи между объектами соответствующих классов.

Отношение зависимости

Отношение зависимости в общем случае указывает некоторое семантическое отношение между двумя элементами модели или двумя множествами таких элементов, которое не является отношением ассоциации, обобщения или реализации. Оно касается только самих элементов модели и не требует множества отдельных примеров для пояснения своего смысла. Отношение зависимости используется в такой ситуации, когда некоторое изменение одного элемента модели может потребовать изменения другого зависимого от него элемента модели.

Отношение ассоциации

Отношение ассоциации соответствует наличию некоторого отношения между классами. Наиболее простой случай данного отношения - бинарная ассоциация. Она связывает в точности два класса и, как исключение, может связывать класс с самим собой. Для бинарной ассоциации на диаграмме может быть указан порядок следования классов с использованием треугольника в форме стрелки рядом с именем данной ассоциации. Направление этой стрелки указывает на порядок классов, один из которых является первым (со стороны треугольника), а другой - вторым (со стороны вершины треугольника). Отсутствие данной стрелки рядом с именем ассоциации означает, что порядок следования классов в рассматриваемом отношении не определен.

Отношение агрегации

Отношение агрегации имеет место между несколькими классами в том случае, если один из классов представляет собой некоторую сущность, включающую в себя в качестве составных частей другие сущности.

Данное отношение имеет фундаментальное значение для описания структуры сложных систем, поскольку применяется для представления системных взаимосвязей типа "часть-целое". Раскрывая внутреннюю структуру системы, отношение агрегации показывает, из каких компонентов состоит система и как они связаны между собой. С точки зрения модели отдельные части системы могут выступать как в виде элементов, так и в виде подсистем, которые, в свою

очередь, тоже могут образовывать составные компоненты или подсистемы. Это отношение по своей сути описывает декомпозицию или разбиение сложной системы на более простые составные части, которые также могут быть подвергнуты декомпозиции, если в этом возникнет необходимость в последующем.

Отношение композиции

Отношение композиции является частным случаем отношения агрегации. Это отношение служит для выделения специальной формы отношения "часть-целое", при которой составляющие части в некотором смысле находятся внутри целого. Специфика взаимосвязи между ними заключается в том, что части не могут выступать в отрыве от целого, т. е. с уничтожением целого уничтожаются и все его составные части.

Отношение обобщения

Отношение обобщения является обычным таксономическим отношением между более общим элементом (родителем или предком) и более частным или специальным элементом (дочерним или потомком). Данное отношение может использоваться для представления взаимосвязей между пакетами, классами, вариантами использования и другими элементами языка UML.

Применительно к диаграмме классов данное отношение описывает иерархическое строение классов и наследование их свойств и поведения. При этом предполагается, что класс-потомок обладает всеми свойствами и поведением класса-предка, а также имеет свои собственные свойства и поведение, которые отсутствуют у класса-предка.

Объекты

Объект (object) является отдельным экземпляром класса, который создается на этапе выполнения программы. Он имеет свое собственное имя и конкретные значения атрибутов. В силу самых различных причин может возникнуть необходимость показать взаимосвязи не только между классами модели, но и между отдельными объектами, реализующими эти классы. В данном случае может быть разработана диаграмма объектов, которая, хотя и не является канонической в метамодели языка UML, но имеет самостоятельное назначение.

Лекция 9 (4 часа)

ДИАГРАММА СОСТОЯНИЙ (STATECHART DIAGRAM)

Для описания реакции объекта на подобные внешние воздействия и используются диаграммы состояний.

Главное предназначение этой диаграммы - описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели в течение его жизненного цикла.

Диаграмма состояний по существу является графом специального вида, который представляет некоторый автомат. Вершинами этого графа являются состояния и некоторые другие типы элементов автомата (псевдосостояния), которые изображаются соответствующими графическими символами. Дуги графа служат для обозначения переходов из состояния в состояние. Диаграммы состояний могут быть вложены друг в друга, образуя вложенные диаграммы более детального представления отдельных элементов модели.

Автоматы

Автомат (state machine) в языке UML представляет собой некоторый формализм для моделирования поведения элементов модели и системы в целом. В UML автомат является пакетом, в котором определено множество понятий, необходимых для представления поведения моделируемой сущности в виде дискретного пространства с конечным числом состояний и переходов. С другой стороны, автомат описывает поведение отдельного объекта в форме последовательности состояний, которые охватывают все этапы его жизненного цикла, начиная от создания объекта и заканчивая его уничтожением. Каждая диаграмма состояний представляет некоторый автомат.

Основными понятиями, входящими в формализм автомата, являются состояние и переход.

В общем случае автомат представляет динамические аспекты моделируемой системы в виде ориентированного графа, вершины которого соответствуют состояниям, а дуги - переходам. При этом поведение моделируется как после-

довательное перемещение по графу состояний от вершины к вершине по связывающим их дугам с учетом их ориентации. Для графа состояний системы можно ввести в рассмотрение специальные свойства.

1. В языке UML понятие автомата дополнено специальной семантикой входящих в соответствующий пакет элементов.

Правила поведения объекта, моделируемого некоторым автоматом, определяются, с одной стороны, общим формализмом автомата, а с другой - его графическим изображением в языке UML в форме конкретной диаграммы состояний.

Состояние

Понятие состояния (state) является фундаментальным не только в метамоде-ли языка UML, но и в прикладном системном анализе. Вся концепция динамической системы основывается на понятии состояния системы. Однако семантика состояния в языке UML имеет целый ряд специфических особенностей.

В языке UML под состоянием понимается абстрактный метакласс, используемый для моделирования отдельной ситуации, в течение которой имеет место выполнение некоторого условия. Состояние может быть задано в виде набора конкретных значений атрибутов класса или объекта, при этом изменение их отдельных значений будет отражать изменение состояния моделируемого класса или объекта.

Начальное состояние

Начальное состояние представляет собой частный случай состояния, которое не содержит никаких внутренних действий (псевдосостояния). В этом состоянии находится объект по умолчанию в начальный момент времени. Оно служит для указания на диаграмме состояний графической области, от которой начинается процесс изменения состояний.

Конечное состояние

Конечное (финальное) состояние представляет собой частный случай состояния, которое также не содержит никаких внутренних действий (псевдосос-

тояния). В этом состоянии будет находиться объект по умолчанию после завершения работы автомата в конечный момент времени. Оно служит для указания на диаграмме состояний графической области, в которой завершается процесс изменения состояний или жизненный цикл данного объекта.

Переход

Простой переход (simple transition) представляет собой отношение между двумя последовательными состояниями, которое указывает на факт смены одного состояния другим. Пребывание моделируемого объекта в первом состоянии может сопровождаться выполнением некоторых действий, а переход во второе состояние будет возможен после завершения этих действий, а также после удовлетворения некоторых дополнительных условий. В этом случае говорят, что переход срабатывает, Или происходит срабатывание перехода. До срабатывания перехода объект находится в предыдущем от него состоянии, называемым исходным состоянием, или в источнике (не путать с начальным состоянием - это разные понятия), а после его срабатывания объект находится в последующем от него состоянии (целевом состоянии).

Переход осуществляется при наступлении некоторого события: окончания выполнения деятельности (do activity), получении объектом сообщения или приемом сигнала.

Составное состояние и подсостояние

Составное состояние (composite state) - такое сложное состояние, которое состоит из других вложенных в него состояний. Последние будут выступать 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.

Составное состояние может содержать два или более параллельных подавтомата или несколько последовательных подсостояний. Каждое сложное состояние может уточняться только одним из указанных способов. При этом любое из подсостояний, в свою очередь, может являться составным состоянием и

содержать внутри себя другие вложенные подсостояния. Количество уровней вложенности составных состояний не фиксировано в языке UML.

Лекция 10 (4 часа)

ДИАГРАММА ДЕЯТЕЛЬНОСТИ (ACTIVITY DIAGRAM)

При моделировании поведения проектируемой или анализируемой системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической и логической реализации выполняемых системой операций.

Для моделирования процесса выполнения операций в языке UML используются так называемые 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 семантике состояний, которые используются для представления не деятельностей, а действий, и 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 этой, операции в предыдущем состоянии. Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия, а дугами - переходы от одного состояния действия к другому.

Таким образом, диаграммы деятельности можно считать частным случаем диаграмм состояний. Именно они позволяют реализовать в языке UML особенности процедурного и синхронного управления, обусловленного завершением внутренних деятельностей и действий. Основным направлением использования диаграмм деятельности является визуализация особенностей реализации операций классов, когда необходимо представить алгоритмы их выполнения. При этом каждое состояние может являться выполнением операции некоторого класса либо ее части, позволяя использовать диаграммы деятельности для описания реакций на внутренние события системы.

В контексте языка UML деятельность (activity) представляет собой некоторую совокупность отдельных вычислений, выполняемых автоматом. 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. Сам же результат может привести к изменению состояния системы или возвращению некоторого значения.

Состояние действия

Состояние действия (action state) является специальным случаем состояния с некоторым входным действием и по крайней мере одним выходящим из состояния переходом. Этот переход неявно предполагает, что входное действие уже завершилось. Состояние действия не может иметь внутренних переходов, поскольку оно является элементарным. Обычное использование состояния действия заключается в моделировании одного шага выполнения алгоритма (процедуры) или потока управления.

Переходы

При построении диаграммы деятельности используются только нетриггерные переходы, т. е. такие, которые срабатывают сразу после завершения деятельности или выполнения соответствующего действия. Этот переход переводит деятельность в последующее состояние сразу, как только закончится действие в предыдущем состоянии. 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, то сработать может только один из них.

Дорожки

Диаграммы деятельности могут быть использованы не только для спецификации алгоритмов вычислений или потоков управления в программных сис-

темах. Не менее важная область их применения связана с моделированием бизнес-процессов.

Для моделирования особенностей в языке UML используется специальная конструкция, получившее название дорожки (swimlanes). При этом все состояния действия на диаграмме деятельности делятся на отдельные группы, которые отделяются друг от друга вертикальными линиями. Две соседние линии и образуют дорожку, а группа состояний между этими линиями выполняется отдельным подразделением (отделом, группой, отделением, филиалом) компании.

Объекты

В общем случае действия на диаграмме деятельности выполняются над теми или иными объектами. Эти объекты либо инициируют выполнение действий, либо определяют некоторый результат этих действий. При этом действия специфицируют вызовы, которые передаются от одного объекта графа деятельности к другому. Поскольку в таком ракурсе объекты играют определенную роль в понимании процесса деятельности, иногда возникает необходимость явно указать их на диаграмме деятельности.

Лекция 11 (4 часа)

ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ (SEQUENCE DIAGRAM)

Одной из характерных особенностей систем различной природы и назначения является взаимодействие между собой отдельных элементов, из которых образованы эти системы.

В языке UML взаимодействие элементов рассматривается в информационном аспекте их коммуникации, т. е. взаимодействующие объекты обмениваются между собой некоторой информацией. При этом информация принимает форму законченных сообщений. Другими словами, хотя сообщение и имеет информационное содержание, оно приобретает дополнительное свойство оказывать направленное влияние на своего получателя.

Для моделирования взаимодействия объектов в языке UML используются соответствующие диаграммы взаимодействия.

Объекты

На диаграмме последовательности изображаются исключительно те объекты, которые непосредственно участвуют во взаимодействии и не показываются возможные статические ассоциации с другими объектами. 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.

Второе измерение диаграммы последовательности - вертикальная временная ось, направленная сверху вниз. Начальному моменту времени соответствует самая верхняя часть диаграммы. При этом взаимодействия объектов реализуются посредством сообщений, которые посылаются одними объектами другим. Сообщения изображаются в виде горизонтальных стрелок с именем сообщения и также образуют порядок по времени своего возникновения. Другими словами, сообщения, расположенные на диаграмме последовательности выше, инициируются раньше тех, которые расположены ниже. При этом масштаб на оси времени не указывается, поскольку диаграмма последовательности моделирует лишь временную упорядоченность взаимодействий типа "раньше-позже".

Линия жизни объекта

Линия жизни служит для обозначения периода времени, в течение которого объект существует в системе и, следовательно, может потенциально участвовать во всех ее взаимодействиях. Если объект существует в системе постоянно, то и его линия жизни должна продолжаться по всей плоскости диаграммы последовательности от самой верхней ее части до самой нижней.

Отдельные объекты, выполнив свою роль в системе, могут быть уничтожены (разрушены), чтобы освободить занимаемые ими ресурсы. Для таких объектов линия жизни обрывается в момент его уничтожения.

Фокус управления

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, в языке UML применяется специальное понятие, получившее название фокуса управления (focus of control).

В отдельных случаях инициатором взаимодействия в системе может быть актер или внешний пользователь.

Иногда некоторый объект может инициировать рекурсивное взаимодействие с самим собой.

Сообщения

Цель взаимодействия в контексте языка UML заключается в том, чтобы специфицировать коммуникацию между множеством взаимодействующих объектов. Каждое взаимодействие описывается совокупностью сообщений, которыми участвующие в нем объекты обмениваются между собой. В этом смысле сообщение (message) представляет собой законченный фрагмент информации, который отправляется одним объектом другому. При этом прием сообщения инициирует выполнение определенных действий, направленных на решение отдельной задачи тем объектом, которому это сообщение отправлено.

Таким образом, сообщения не только передают некоторую информацию, но и требуют или предполагают от принимающего объекта выполнения ожидаемых действий. Сообщения могут инициировать выполнение операций объектом соответствующего класса, а параметры этих операций передаются вместе с сообщением. На диаграмме последовательности все сообщения упорядочены по времени своего возникновения в моделируемой системе.

Лекция 12 (4 часа)

ДИАГРАММА КООПЕРАЦИИ (COLLABORATION DIAGRAM)

Особенности взаимодействия элементов моделируемой системы могут быть представлены на диаграммах последовательности и кооперации. Если первая служит для визуализации временных аспектов взаимодействия, то диаграмма кооперации предназначена для спецификации структурных аспектов взаимодействия. Главная особенность диаграммы кооперации заключается в возможности графически представить не только последовательность взаимодействия, но и все структурные отношения между объектами, участвующими в этом взаимодействии.

Прежде всего, на диаграмме кооперации в виде прямоугольников изображаются участвующие во взаимодействии объекты, содержащие имя объекта, его класс и, возможно, значения атрибутов. Далее, как и на диаграмме классов, указываются ассоциации между объектами в виде различных соединительных линий. При этом можно явно указать имена ассоциации и ролей, которые играют объекты в данной ассоциации. Дополнительно могут быть изображены динамические связи - потоки сообщений. Они представляются также в виде соединительных линий между объектами, над которыми располагается стрелка с указанием направления, имени сообщения и порядкового номера в общей последовательности инициализации сообщений.

В отличие от диаграммы последовательности, на диаграмме кооперации изображаются только отношения между объектами, играющими определенные роли во взаимодействии. С другой стороны, на этой диаграмме не указывается время в виде отдельного измерения. Поэтому последовательность взаимодействий и параллельных потоков может быть определена с помощью порядковых номеров. Следовательно, если необходимо явно специфицировать взаимосвязи между объектами в реальном времени, лучше это делать на диаграмме последовательности.

Поведение системы может описываться на уровне отдельных объектов, которые обмениваются между собой сообщениями, чтобы достичь нужной цели

или реализовать некоторый сервис. С точки зрения аналитика или конструктора важно представить в проекте системы структурные связи отдельных объектов между собой. Такое статическое представление структуры системы как совокупности взаимодействующих объектов и обеспечивает диаграмма кооперации.

Таким образом, с помощью диаграммы кооперации можно описать полный контекст взаимодействий как своеобразный временной "среза" совокупности объектов, взаимодействующих между собой для выполнения определенной задачи или бизнес-цели программной системы.

IV. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ПРОВЕДЕНИЮ И ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

Лабораторные работы проводятся по подгруппам в компьютерном классе. Каждый студент получает задание. Выполняя задание студент пользуется материалом изложенным в лабораторной работе и изученным на лекциях.

V. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ОРГАНИЗАЦИИ МЕЖСЕССИОННОГО КОНТРОЛЯ ЗНАНИЙ СТУДЕНТОВ

Межсессионный контроль осуществляется во время проведения лабораторных работ посредством приема отчетов, устного опроса. По итогам выполнения перечисленных видов работ преподавателем выставляется аттестационная оценка по пятибалльной системе.

VI. ЗАДАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ

Лабораторная работа № 1 (2 часа)

ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ В ПРЕДМЕТНУЮ ОБЛАСТЬ

Цель работы: Ознакомиться с системой «Служба занятости в рамках вуза».

Процесс создания диаграмм начинается с этапа изучения предметной области.

Задание: Изучить предметную область системы «Служба занятости в рамках вуза».

Лабораторная работа № 2 (2 часа)

МЕТОДОЛОГИЯ IDEF0

Цель работы:

- изучение основных принципов методологии IDEF0,
- создание нового проекта в BPWin,
- формирование контекстной диаграммы,
- проведение связей.

Задание:

1. Создать новый проект в BPWin.
2. Сформировать контекстную диаграмму по системе согласно методологии IDEF0.
3. Задать входы, выходы, механизмы и управление.
4. Декомпозировать контекстную диаграмму.
5. Провести связи по выходу.
6. Провести связи по управлению.
7. Провести связи по входу
8. Сохранить проект в отдельный файл.

Лабораторная работа № 3 (2 часа)

ДОПОЛНЕНИЕ МОДЕЛЕЙ ПРОЦЕССОВ ДИАГРАММАМИ

DFT и DFT и WorkFlow (IDFT3)

Цель работы:

• построение диаграмм потоков данных (DFT),

• описание взаимосвязей между процессами при помощи диаграмм IDET3 (WorkFlow).

Задания:

1. Дополнить созданную диаграмму IDEFO диаграммой DFD.
2. Добавить на диаграмму DFD внешнюю сущность и хранилище данных.
3. Связать диаграмму и внешнюю сущность.
4. Связать диаграмму и хранилище.
5. Определить имя связи с внешней сущностью.
6. Создать диаграмму IDEF3, определяющую последовательность заполнения БД системы.
7. Связать работы на диаграмме.
8. Добавить на диаграмму перекрестки, моделирующие параллельные события при заполнении БД.
9. Добавить объект-ссылку и связать его с диаграммой.

Лабораторная работа № 4 (2 часа)

ОТЧЕТЫ в BPWin

Цель работы:

• изучить виды отчетов и способы их создания;

• освоить метод поиска ошибок в диаграммах, используя отчеты.

Задания:

1. Создать отчет по модели по диаграмме IDEFO, созданной в первой лабораторной работе.
2. Сохранить отчет в файл.

3. Открыть диалоговое окно отчета по стрелкам и сформировать в нем стандартный отчет, содержащий информацию о началах и концах стрелок.

4. Сохранить стандартный отчет под именем Arrows Source/Dest.

5. Создать отчет согласованности с методологией.

6. Сохранить полученный отчет в файл.

7. Проверить отчет на наличие сообщений об ошибках в модели.

Лабораторная работа № 5 (2 часа)

ДИАГРАММЫ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

Цель работы:

•изучение диаграмм вариантов использования,

•изучение их применения в процессе постановки задачи.

Задания:

1.Определять основные функции системы.

2.Для нескольких функций выделить действующие лица, участвующие в них или использующие их результаты.

3.Для каждой выбранной функции построить диаграмму вариантов использования.

Лабораторная работа № 6 (2 часа)

ДИАГРАММЫ КЛАССОВ

Цель работы:

•изучение диаграмм классов,

•изучение их применения в процессе проектирования.

Задания:

1. Выделить основные классы объектов к проектируемой системе.

2. Построить диаграмму классов, в общем виде демонстрирующую архитектуру системы.

3. Построить одну-две диаграммы классов, детализирующие отдельные подсистемы. Указать для классов основные атрибуты и операции, указать вид и направление ассоциаций.

Лабораторная работа № 7 (2 часа)

ДИАГРАММЫ ВЗАИМОДЕЙСТВИЯ

Цель работы:

• изучение диаграмм взаимодействия,

• изучение их применения в процессе проектирования.

Задания:

1. Выбрать в моделируемой системе вариант использования, для которого будут строиться диаграммы взаимодействия.

2. Построить для выбранного варианта использования диаграмму последовательности.

3. Построить для того же варианта использования кооперативную диаграмму.

4. Сформулировать достоинства и недостатки каждого вида диаграмм при моделировании данного варианта использования.

Лабораторная работа № 8 (2 часа)

ДИАГРАММЫ СОСТОЯНИЙ

Цель работы:

1. изучение диаграмм состояний,

2. изучение их применения в процессе проектирования.

Задания:

1. Выбрать и моделируемой системе классы, для объектов которых будут строиться диаграммы состояний.

2. Построить для каждого выбранного класса диаграмму состояний, характеризующую поведение его объектов в нескольких вариантах использования.

Лабораторная работа № 9 (2 часа)

ДИАГРАММЫ ПАКЕТОВ, КОМПОНЕНТОВ И РАЗМЕЩЕНИЯ

Цель работы:

1. изучение диаграмм пакетов, диаграммы компонентов и диаграммы размещения,

2. изучение их применения в процессе проектирования.

Задания:

1. Построить для моделируемой системы общую диаграмму пакетов, отметить на ней пакеты с необходимыми системными библиотеками, отобразить зависимости между пакетами.

2. Построить для данной системы диаграмму компонентов, соответствующую построенной диаграмме пакетов, системные пакеты изобразить в виде спецификаций пакетов,

3. Построить для проектируемой системы несколько вариантов диаграммы размещения (для архитектуры «клиент-сервер», трехуровневой архитектуры и т. д.), обосновать каждый вариант, предложить наиболее оптимальный.

VII. КОМПЛЕКТЫ ЭКЗАМЕНАЦИОННЫХ БИЛЕТОВ

Экзаменационный билет № 1

1. Диаграмма IDEF0
2. Диаграмма вариантов использования

Экзаменационный билет № 2

1. Стрелки в IDEF0
2. Вариант использования

Экзаменационный билет № 3

1. Диаграммы и работы в IDEF3
2. Актер на диаграмме вариантов использования

Экзаменационный билет № 4

1. Язык UML
2. Интерфейсы на диаграмме вариантов использования

Экзаменационный билет № 5

1. Назначение языка UML
2. Отношения на диаграмме вариантов использования

Экзаменационный билет № 6

1. Диаграмма DFD
2. Диаграмма классов

Экзаменационный билет № 7

1. Стрелки, хранилища, работы в диаграмме DFD
2. Класс на диаграмме классов

Экзаменационный билет № 8

1. Диаграмма IDEF0
2. Отношения между классами

Экзаменационный билет № 9

1. Стрелки в IDEF0
2. Диаграмма состояний

Экзаменационный билет № 10

1. Связи в IDEF0
2. Автоматы на диаграмме состояний

Экзаменационный билет № 11

1. Диаграмма DFD
2. Состояния на диаграмме состояний

Экзаменационный билет № 12

1. Стрелки, хранилища, работы в диаграмме DFD
2. Переходы, сложные переходы

Экзаменационный билет № 13

1. Метод описания процессов IDEF3
2. Диаграмма деятельности

Экзаменационный билет № 14

1. Диаграммы и работы в IDEF3
2. Состояние действия

Экзаменационный билет № 15

1. Связи в IDEF3
2. Переходы

Экзаменационный билет № 16

1. Перекрестки
2. Дорожки на диаграмме деятельности

Экзаменационный билет № 17

1. Объект ссылки
2. Объекты на диаграмме деятельности

Экзаменационный билет № 18

1. Декомпозиция работ
2. Отношения на диаграмме вариантов использования

Экзаменационный билет № 19

1. Создание смешанной модели
2. Отношения между классами

Экзаменационный билет № 20

1. Стрелки, хранилища, работы в диаграмме DFD
2. Диаграмма классов

VIII. КАРТА ОБЕСПЕЧЕННОСТИ ДИСЦИПЛИНЫ КАФЕДРАМИ ПРОФЕССОРСКО-ПРЕПОДАВАТЕЛЬСКОГО СОСТАВА

Все виды занятий по дисциплине ведет ассистент Жилиндина О.В.

СОДЕРЖАНИЕ

Ольга Викторовна Жилиндина,
ассистент кафедры ИиУС АмГУ

Учебно-методический комплекс по дисциплине «Элементы моделирования и проектирования информационных систем»

Изд-во АмГУ. Подписано к печати Формат 60x84/16. Усл. печ. л .
Тираж Заказ