Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования «Амурский государственный университет»

Кафедра Информационных и управляющих систем
(наименование кафедры)

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ДИСЦИПЛИНЫ

Методология разработки корпоративных приложений
(наименование дисциплины)

Основной образовательной программы по направлению подготовки:

231000.68 «Программная инженерия»
по магистерской программе «Управление разработкой программного обеспечения»
(код и наименование направления)

Благовещенск 2011

УМКД разработан _____
(степень, звание, фамилия, имя, отчество разработчиков)
_____
_____

Рассмотрен и рекомендован на заседании кафедры

Протокол заседания кафедры от «____» _____ 201___ г. № _____

Зав. кафедрой _____ /_____ /
(подпись) (И.О. Фамилия)

УТВЕРЖДЕН

Протокол заседания УМСС _____
(указывается название специальности (направления подготовки))

от «____» _____ 201___ г. № _____

Председатель УМСС _____ /_____ /
(подпись) (И.О.Фамилия)

1. Рабочая программа учебной дисциплины

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Амурский государственный университет»

УТВЕРЖДАЮ
Проректор по учебной работе
_____ В.В. Проказин
«___» _____ 20__ г.

РАБОЧАЯ ПРОГРАММА
МЕТОДОЛОГИЯ РАЗРАБОТКИ КОРПОРАТИВНЫХ ПРИЛОЖЕНИЙ

Направление подготовки 231000.68 «Программная инженерия»
по магистерской программе «Управление разработкой программного обеспечения»
Квалификация (степень) выпускника – магистр
Специальное звание – нет

Курс – 2	Семестр – 3
Лекции – 0 (час.)	Экзамен – 3
Практические (семинарские) занятия – 36 (час.)	Зачет – нет

Лабораторные занятия – 18 (час.)
Самостоятельная работа – 90 (час.)
Общая трудоемкость дисциплины – 180 (час.), 5 (з.е.)
Курсовая работа (проект) – нет
Составитель – А.В. Бушманов, доцент, канд. техн. наук
Факультет математики и информатики
Кафедра информационных и управляющих систем

2012 г.

Рабочая программа составлена на основании Федерального государственного образовательного стандарта высшего профессионального образования по направлению подготовки 231000 «Программная инженерия» (квалификация (степень) «магистр»)

Рабочая программа обсуждена на заседании кафедры информационных и управляющих систем
«___» _____ 20___ г., протокол № _____
Заведующий кафедрой _____ А.В. Бушманов

Рабочая программа одобрена на заседании учебно-методического совета направления подготовки 231000.68 «Программная инженерия»
«___» _____ 20___ г., протокол № _____
Председатель _____ В.В. Еремина

Рабочая программа переутверждена на заседании кафедры информационных и управляющих систем
«___» _____ 20___ г., протокол № _____
Заведующий кафедрой _____ А.В. Бушманов

СОГЛАСОВАНО
Начальник учебно-методического управления
_____
«___» _____ 20___ г.

СОГЛАСОВАНО
Председатель учебно-методического совета факультета
_____ С.Г. Самохвалова
«___» _____ 20___ г.

СОГЛАСОВАНО
Заведующий выпускающей кафедрой
_____ А.В. Бушманов
«___» _____ 20___ г.

СОГЛАСОВАНО
Директор научной библиотеки
_____ Л.А. Проказина
«___» _____ 20___ г.

1 ЦЕЛИ И ЗАДАЧИ ОСВОЕНИЯ ДИСЦИПЛИНЫ

Ознакомление студентов с современными технологиями проектирования и создания корпоративных информационных систем: от управленческих задач к инструментам и средствам их решения. Вооружить специалистов разработчиков информационных систем организаций, корпораций и холдингов конкретными знаниями, умениями и навыками основ проектирования корпоративных информационных систем производственных предприятий

Задачами изучения дисциплины является приобретение студентами теоретических знаний и представлений о принципах разработки, используемых технологий при проектировании современных КИС.

2 МЕСТО ДИСЦИПЛИНЫ В СТРУКТУРЕ ООП ВПО

Дисциплина относится к базовой части профессионального цикла (М 2.Б.1) Федерального государственного образовательного стандарта высшего профессионального образования по направлению подготовки 231000.68 «Программная инженерия» (квалификация (степень) «магистр»).

Для успешного освоения данной дисциплины необходимы знания, умения и навыки, приобретенные в результате освоения дисциплин направления Федерального государственного образовательного стандарта высшего профессионального образования по направлению подготовки 231000 «Программная инженерия» (квалификация (степень) «бакалавр»): «Метрология, стандартизация и сертификация», «Информационные технологии», «Корпоративные информационные системы».

Знания, умения и навыки, приобретенные в результате освоения данной дисциплины необходимы для освоения дисциплин базовой и вариативной части профессионального цикла (М.2) Федерального государственного образовательного стандарта высшего профессионального образования по направлению подготовки 231000 «Программная инженерия» (квалификация (степень) «магистр»), а также прохождения научно-исследовательской практики и выполнения научно-исследовательской работы (М.3).

3 КОМПЕТЕНЦИИ ОБУЧАЮЩЕГОСЯ, ФОРМИРУЕМЫЕ В РЕЗУЛЬТАТЕ ОСВОЕНИЯ ДИСЦИПЛИНЫ

В результате освоения дисциплины обучающийся должен демонстрировать следующие результаты образования:

1) Знать: состав и структуру различных классов КИС, современные технологии разработки проектирования КИС, содержание этапов разработки БИС, методы и инструментальные средства разработки КИС.

2) Уметь: выбирать способы формализации процессов разработки КИС, состав и содержание технологических операций разработки КИС, разрабатывать планы выполнения проектировочных работ, ставить и решать задачи информационного обеспечения процесса управления.

3) Владеть: навыками логико-методологического анализа научного исследования и его результатов.

В процессе освоения данной дисциплины студент формирует и демонстрирует следующие общекультурные и профессиональные компетенции:

способность рассчитывать и оценивать условия и последствия принимаемых организационно-управленческих решений (ПК-14);

навыки подготовки и проведения учебных занятий по дисциплинам направления «Программная инженерия» (ПК-13);

умение применять современные технологии разработки программных комплексов с использованием автоматизированных систем планирования и управления, осуществлять контроль качества программных продуктов (ПК-9).

4 СТРУКТУРА И СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

Общая трудоемкость дисциплины составляет 5 зачетных единиц, 180 час.

№ п/п	Раздел дисциплины	Семестр	Неделя семестра	Виды учебной работы, включая самостоятельную работу студентов и трудоемкость в часах				Формы текущего контроля успеваемости Форма промежуточной аттестации
				Лек	Пр	Лаб	Сам	
1	Философия и методология проектирования.	3	1	-	4	-	10	Собеседование
2	Метод COMET (Concurrent Object Modeling and Architectural Design Method).	3	2	-	4	2	10	Семинар Лабораторная раб.
3	Проектирование информационной модели.	3	3	-	4	2	10	Семинар Лабораторная раб.
4	Проектирование информационной системы.	3	4	-	4	2	10	Семинар Лабораторная раб.
5	Тестирование информационной системы.	3	5	-	4	2	10	Семинар Лабораторная раб.
6	Системное программное обеспечение.	3	6	-	4	2	10	Семинар Лабораторная раб.
7	Прикладное программное обеспечение.	3	7	-	4	4	16	Семинар Лабораторная раб.
8	Интеграция программного обеспечения.	3	8-9	-	8	4	14	Семинар Лабораторная раб.
9	Всего по разделам	3	1-18	-	36	18	90	Зачет

5 СОДЕРЖАНИЕ РАЗДЕЛОВ И ТЕМ ДИСЦИПЛИНЫ

5.1 Практические занятия

5.1.1 Практическое занятие 1. Философия и методология проектирования.

Определение контекста, значения, входных и выходных информационных потоков предметной области «проектирование информационных систем (ИС)». Эволюция предметной области «проектирование ИС». Определение понятия «метода» проектирования и «нотации»; обзор наиболее значимых (распространенных) методов и нотаций. Определение понятий «информационная система», «информационная модель», «жизненный цикл ИС», «роль». Типовые роли, задействованные в процессе проектирования ИС.

5.1.2 Практическое занятие 2. Метод COMET (Concurrent Object Modeling and Architectural Design Method).

Назначение и концепции метода, задействованные роли, жизненный цикл разработки, понятия объекта, класса, операции, задачи. Фазы проектирования информационной модели и программной системы; связь между фазами. Определение понятий «прецедент», «ассоциация», «сущность», «событие», «сообщение», «контекст модели», «контекст системы», «архитектурный стиль».

5.1.3 Практическое занятие 3. Проектирование информационной модели.

Моделирование прецедентов, их выявление и документирование. Статическое моделирование: ассоциации между классами; иерархии композиции/агрегирования и обобще-

ния/специализации; выявление и документирование сущностных классов; критерии выявления; контекст модели.

5.1.4 Практическое занятие 4. Проектирование информационной системы.

Разбиение на классы и объекты; критерии разбиения, категории классов, подсистемы. Конечные автоматы и диаграммы состояний: события, состояния, условия переходов, плоские, иерархические и параллельные диаграммы состояний. Динамическое моделирование: взаимодействие объектов, диаграммы взаимодействия, динамический анализ. Проектирование архитектуры системы: существующие архитектурные стили, декомпозиция, критерии декомпозиции, разделение обязанностей, консолидированные диаграммы кооперации. Проектирование архитектуры распределенных систем: конфигурируемые архитектуры, декомпозиция, критерии декомпозиции, интерфейсы подсистем, управление транзакциями, распределение данных. Разбиение на задачи: критерии разбиения, инверсия и архитектура задач, межзадачные коммуникации, спецификации поведения задач. Проектирование классов: классы, скрывающие данные, алгоритмы, бизнес-логику, интерфейс пользователя; применение наследования в проектировании классов; спецификации интерфейсов классов. Детальное проектирование: составные задачи, синхронизация доступа, «разъемы» для межзадачных коммуникаций, логика упорядочения событий. Анализ производительности: теория планирования в реальном времени, анализ с помощью последовательности событий, смешанный анализ; пересмотр проекта; оценка и изменение параметров производительности.

5.1.5 Практическое занятие 5. Тестирование информационной системы.

Типы тестирования. Место тестирования в жизненном цикле разработки ИС. Типовые роли, участвующие в процессе тестирования. Инструментальные средства тестирования.

5.1.6 Практическое занятие 6. Системное программное обеспечение.

Существующие типы системного ПО. Место системного ПО в жизненном цикле предприятия. Операционные системы: назначение, архитектура, функции. Системы управления базами данных (СУБД): назначение, архитектура, функции. Программное обеспечение «среднего» слоя: назначение, архитектура, функции. Эксплуатация системного ПО; задачи персонала по эксплуатации.

5.1.7 Практическое занятие 7. Прикладное программное обеспечение.

Существующие типы прикладного ПО. Место прикладного ПО в жизненном цикле предприятия. Корпоративные информационные системы: назначение, архитектура, функции. Эксплуатация прикладного ПО; задачи персонала по эксплуатации.

5.1.8 Практическое занятие 8. Интеграция программного обеспечения.

Существующие типы интеграции ПО.

5.1.9 Место интеграции ПО в жизненном цикле предприятия.

5.2 Лабораторные занятия.

5.2.1 Лабораторная работа 1. Интегрированные распределенные приложения.

5.2.2 Лабораторная работа 2. Протокол ODBC и его реализации.

5.2.3 Лабораторная работа 3 Язык SQL - базовый интерфейс SQL-сервера.

5.2.4 Лабораторная работа 4. Физическое проектирование баз данных.

5.2.5 Лабораторная работа 5. Язык программирования Java.

5.2.6 Лабораторная работа 6. Программное обеспечение «среднего» слоя: назначение, архитектура, функции.

5.2.7 Лабораторная работа 7. Информационные приложения, основанные на использовании "складов данных" (DataWarehousing).

5.2.8 Лабораторная работа 8. Проблема "унаследованных систем" (Legacy Systems).

6 САМОСТОЯТЕЛЬНАЯ РАБОТА

№ п/п	Раздел дисциплины	Форма (вид) самостоятельной работы	Трудоемкость в часах
1	Философия и методология	Оформление отчета.	10

	проектирования.		
2	Метод COMET (Concurrent Object Modeling and Architectural Design Method).	Оформление отчета.	10
3	Проектирование информационной модели.	Оформление отчета.	10
4	Проектирование информационной системы.	Оформление отчета.	12
5	Тестирование информационной системы.	Оформление отчета.	12
6	Системное программное обеспечение.	Оформление отчета.	12
7	Прикладное программное обеспечение.	Оформление отчета.	12
8	Интеграция программного обеспечения.	Оформление отчета.	12

7 МАТРИЦА КОМПЕТЕНЦИЙ

№ п/п	Раздел дисциплины	Общее кол-во компетенций			
		ПК-9	ПК-13	ПК-14	
1	Природа научного познания.	+			1
2	Характер научного знания и его функции.	+			1
3	Схема научного исследования.		+	+	2
4	Методологические основы исследования.		+	+	2
5	Программная инженерия как отрасль научного знания.		+	+	2

8 ОБРАЗОВАТЕЛЬНЫЕ ТЕХНОЛОГИИ

Образовательный процесс по дисциплине строится на основе комбинации следующих образовательных технологий.

Интегральную модель образовательного процесса по дисциплине формируют технологии методологического уровня: модульно-рейтинговое обучение, технология поэтапного формирования умственных действий, технология развивающего обучения, элементы технологии развития критического мышления.

Реализация данной модели предполагает использование следующих технологий стратегического уровня (задающих организационные формы взаимодействия субъектов образовательного процесса), осуществляемых с использованием определенных тактических процедур:

– лекционные (вводная лекция, информационная лекция, обзорная лекция, лекция-консультация, проблемная лекция);

– практические (углубление знаний, полученных на теоретических занятиях, решение задач);

– тренинговые (формирование определенных умений и навыков, формирование алгоритмического мышления);

– активизации познавательной деятельности (приемы технологии развития критического мышления через чтение и письмо, работа с литературой, подготовка презентаций по темам домашних работ);

– самоуправления (самостоятельная работа студентов, самостоятельное изучение материала).

Рекомендуется использование информационных технологий при организации коммуникации со студентами для представления информации, выдачи рекомендаций и консультирования по оперативным вопросам (электронная почта), использование при проведении лекционных и практических занятий мультимедиа-средств.

9 ОЦЕНОЧНЫЕ СРЕДСТВА ДЛЯ ТЕКУЩЕГО КОНТРОЛЯ УСПЕВАЕМОСТИ, ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ ПО ИТОГАМ ОСВОЕНИЯ ДИСЦИПЛИНЫ И УЧЕБНО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

9.1 Оценочные средства для текущего контроля успеваемости.

9.1.1 Контрольные вопросы допуска к выполнению реферативных работ.

9.1.2 Отчеты о выполнении индивидуальных вариантов заданий.

9.2 Оценочные средства для промежуточной аттестации

Вопросы к экзамену:

9.2.1 Общее представление об информационной системе.

9.2.2 Общая классификация архитектур информационных приложений.

9.2.3 . Средства и методологии проектирования, разработки и сопровождения файл-серверных приложений.

9.2.4 Средства и методологии проектирования, разработки и сопровождения клиент-серверных приложений.

9.2.5 Средства и методологии проектирования, разработки и сопровождения Intranet-приложений.

9.2.6 Информационные приложения, основанные на использовании "складов данных" (DataWarehousing).

9.2.7 Глобально распределенные информационные системы.

9.3 Учебно-методическое обеспечение самостоятельной работы

9.3.1 Карточки с заданиями и методическими указаниями по выполнению практических работ

9.3.2 СТО СМК 4.2.3.05-2011. Стандарт ФГБОУОВПО «АмГУ». Оформление выпускных квалификационных и курсовых работ (проектов), 2011. – 95 с.

10 УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

а) основная литература:

10.1 Избачков Ю.С. Информационные системы : учеб.: рек. Мин. обр. и науки РФ/ Ю. С. Избачков, В. Н. Петров. -2-е изд. -СПб.: Питер, 2008.-656 с.

10.2 Гвоздева Т.В. Проектирование информационных систем: учеб. пособие: рек. УМО/ Т.В. Гвоздева, Б.А. Баллод. -Ростов н/Д: Феникс, 2009.-509 с.

10.3 Хомоненко А. Д. Базы данных : учеб.: рек. УМО / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев; под ред. А. Д. Хомоненко. -6-е изд., доп. -СПб.: КОРОНА-Век, 2009.-736 с.

б) дополнительная литература:

10.4 Соловьев И.В. Проектирование информационных систем. Фундаментальный курс: учеб. пособие: рек. УМО/ И. В. Соловьев, А. А. Майоров. -М.: Академический Проект, 2009.-399 с.

10.5 Уткин В.Б. Информационные системы в экономике: учеб.: рек. УМО/ В. Б. Уткин, К. В. Балдин. -5-е изд., стер. -М.: Академия, 2010.-284 с.

10.6 Автоматизированные информационные технологии в экономике: учеб. для вузов: рек. Мин. обр. РФ/ под ред. Г. А. Титоренко. -М.: Юнити, 2000, 2006.-400 с.

10.7 Емельянова Н.З. Основы построения автоматизированных информационных систем: учеб. пособие: рек. Мин. обр. РФ/ Н. З. Емельянова, Т. Л. Партыка, И. И. Попов. -М.: ФОРУМ: ИНФРА-М, 2007.-416 с.

10.8 Исаев Г.Н. Информационные системы в экономике : учеб. : доп. Мин. обр. РФ/ Г.Н. Исаев. -2-е изд., стер.. -М.: Омега-Л, 2009.-463 с.

в) периодические издания:

10.8 Информационные системы и технологии.

10.9 Программные продукты и системы.

10.10 Проблемы передачи информации.

г) программное обеспечение и Интернет-ресурсы:

Свободно распространяемое программное обеспечение

№ п/п	Наименование ресурса	Характеристика
1	рее http://www.iqlib.ru	Интернет библиотека образовательных изданий, в которой собраны электронные учебники, справочные и учебные пособия. Удобный поиск по ключевым словам, отдельным темам и отраслям знаний.
2	http://www.intuit.ru	Интернет-университет информационных технологий, в котором вобраны электронные и видео-курсы по отраслям знаний
3	http://amursu.ru	Сайт АмГУ, Библиотека – электронная библиотека АмГУ
4	http://www.biblioclub.ru	Электронная библиотечная система «Университетская библиотека – online»: специализируется на учебных материалах для ВУЗов по научно-гуманитарной тематике, а так же содержит материалы по точным и естественным наукам

11 МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

11.1 Лекционная аудитория, оборудованная мультимедийными средствами

11.2 Учебные кабинеты, оборудованные рабочими местами пользователей ЭВМ

12 РЕЙТИНГОВАЯ ОЦЕНКА ЗНАНИЙ СТУДЕНТОВ ПО ДИСЦИПЛИНЕ

Семестровый модуль дисциплины						
№ п/п	Раздел дисциплины	Виды контроля	Сроки выполнения (недели)	Максимальное кол-во баллов	Посещение, активность на занятиях	Максимальное кол-во баллов за модуль
1	Философия и методология проектирования.	ПР № 1	1	4	1	7
2	Метод COMET (Concurrent Object Modeling and Architectural Design Method).	ПР № 2 ЛР № 1	1	3	1	7
			1	2	1	

3	Проектирование информационной модели.	ПР № 3 ЛР № 2	1 1	3 2	1 1	7
4	Проектирование информационной системы.	ПР № 4 ЛР № 3	1 1	3 3	1 1	8
5	Тестирование информационной системы.	ПР № 5 ЛР № 4	1 1	3 3	1 1	8
6	Системное программное обеспечение.	ПР № 6 ЛР № 5	1 1	3 3	1 1	8
7	Прикладное программное обеспечение.	ПР № 7 ЛР № 6-7	1 1	3 3	1 1	8
8	Интеграция программного обеспечения.	ПР № 8-9 ЛР № 8-9	1 1	3 4	1 1	7
9	Промежуточная аттестация	зачет	1	6	16	40
Итого						100

2 Краткое изложение программного материала

Практическое занятие 1. Философия и методология проектирования.

В чем заключается суть *практики* и *теории*? Она заключается в том, что теория — это знания, которые можно передать от одного индивида другому, в то время как практика является субъективным опытом, представляющим собой не отчуждаемые знания. Таким образом, общая картина целостного "мира проектирования ПО" диалектична, т.е. представляет собой дихотомию из теоретической и практической информации.

В чем же разница между *философией* и *методологией* проектирования? В теории проектирования нет устоявшегося ответа на этот вопрос, по этому каждый автор, задающийся им, отвечает на него самостоятельно.

Под *философией проектирования* будем понимать такую культуру *рефлексии*, которая основана на логически непротиворечивом рациональном подходе к решению технических вопросов. Это значит, что любой технический вопрос, стоящий перед сознанием специалиста, решается таким образом, что полученное решение является логически непротиворечивым высказыванием, истинным в рамках явных допущений, сделанных с целью ограничения рассматриваемой области решения.

Под *методологией проектирования* будем понимать такие основополагающие знания, которые являются логически замкнутым идейным базисом, не специфичным какой-то очередной технологии. Это значит, что технический специалист, владеющий методологией проектирования и использующий ее в качестве инструмента рефлексии, потенциально способен принимать решения лучшего качества, чем его коллега, не владеющий указанным ментальным инструментом.

Знания, входящие в *методологию проектирования*, руководствуясь критерием сильной/слабой связности можно условно разделить на следующий перечень категорий:

1. Проектные предположения
2. Рассмотрение (идентификация) и постановка задачи
3. Именование, сокрытие информации и повторное использование
4. Декомпозиция и ее критерии
5. Архитектурные стили и программные приемы
6. Изменение систем во времени
7. Оценка архитектуры
8. Программная метрология

Проектные предположения позволяют выделить из непрерывного пространства бытия некоторую область знаний, называемую нами *пространством решений*, которая необходима

для того, чтобы требуемая к выполнению техническая задача была конечна и потенциально разрешима.

Рассмотрение задачи представляет собой ее идентификацию, т.е. отделение информации, связанной с задачей, от информации, с ней не связанной. Это весьма непростая работа, т.к. имеют место два взаимоисключающих действия: с одной стороны, добавление в пространство решений новых деталей означает повышение точности итогового решения, с другой стороны, добавление новых деталей увеличивает итоговую сложность задачи.

Найти оптимальное решение этой головоломки удается тогда, когда удается понять, какое решение для задачи является наиболее приемлемым. После того, как такое решение найдено, остается проверить условия задачи на истинность. Для этого искусственно расширяют заданные на предыдущем этапе границы задачи с целью изучения условий, характерных для новых границ. Если новые условия не вступают в противоречие со старыми — на лицо "истинность" первоначальных условий, в противном случае имеют место ошибки, минимизация влияния которых на идентификацию задачи, — работа системного аналитика.

Ключевые вопросы:

1. Определение контекста, значения, входных и выходных информационных потоков предметной области «проектирование информационных систем (ИС)»;
2. Эволюция предметной области «проектирование ИС»;
3. Определение понятия «метода» проектирования и «нотации»;
4. Особенности современной науки;
5. Обзор наиболее значимых (распространенных) методов и нотаций;
6. Определение понятий «информационная система», «информационная модель», «жизненный цикл ИС», «роль»;
7. Типовые роли, задействованные в процессе проектирования ИС.

Литература:

1 Избачков Ю.С. Информационные системы: учеб.:рек. Мин. обр. и науки РФ/ Ю. С. Избачков, В. Н. Петров. -2-е изд. -СПб.: Питер, 2008.-656 с.

2 Гвоздева Т.В. Проектирование информационных систем: учеб. пособие: рек. УМО/ Т.В. Гвоздева, Б.А. Баллод. -Ростов н/Д: Феникс, 2009.-509 с.

3 Хомоненко А. Д. Базы данных: учеб.: рек. УМО / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев; под ред. А. Д. Хомоненко. -6-е изд., доп. -СПб.: КОРОНА-Век, 2009.-736 с.

Практическое занятие 2. Метод COMET (Concurrent Object Modeling and Architectural Design Method).

Метод OMT оказал большое влияние на разработчиков объектно-ориентированных методов и технологий, внеся, таким образом, значительный вклад в теорию проектирования ИС. Кроме этого, графическая нотация описания моделей, предложенная в нем, широко применяется в других методах, а также в литературе, посвященной объектно-ориентированной разработке программных систем.

В методе OMT проектируемая программная система представляется в виде трех взаимосвязанных моделей:

1. *Объектной модели*, которая определяет статические аспекты системы, в основном связанные с данными (статическая структура);
2. *Динамической модели*, которая описывает работу отдельных частей системы (динамическая структура);
3. *Функциональной модели*, которая описывает взаимодействие отдельных частей системы, возникающее в процессе ее работы (взаимодействие описывается как в разрезе данных, так и в разрезе управления).

Эти три модели позволяют получить три ортогональных представления проектируемой системы в одной нотации. Совокупность моделей системы может быть проинтерпретирована на компьютере (с помощью инструментального ПО), что позволяет продемонстрировать ха-

рактер работы с будущей системой и существенно упрощает согласование предварительного проекта системы.

На основе метода OMT построена технология проектирования, опирающаяся на программный продукт OMTTool, который позволяет разрабатывать модели проектируемой программной системы в интерактивном режиме с использованием многооконного графического редактора и интерпретатора наборов диаграмм, составляемых при анализе требований к системе и ее проектировании. При этом, как только получен достаточно полный набор диаграмм проектируемой программной системы, его можно проинтерпретировать и предварительно оценить различные свойства будущей реализации системы. В настоящее время OMTTool входит в состав системы Paradigm+.

Метод OMT ограничивается двумя фазами жизненного цикла ПО (ИС): анализом требований вкупе с построением объектной модели и реализацией программной системы.

Метод COMeT (Concurrent Object Modeling Technique) основан на методе COMADM (Concurrent Object Modeling and Architectural Design Method), имея с ним незначительные отличия, обусловленные используемой в методе COMET нотации UML. Указанные два метода разработаны профессором кафедры программотехники университета Джорджа Мэйсона Хассаном Гома (Hassan Gomaa), авторитетным идеологом области проектирования распределенных приложений и приложений реального времени.

В общем виде метод COMET представляет собой метод объектного моделирования и архитектурного проектирования параллельных систем, в основе которого лежит создание объектно-ориентированного ПО. Жизненный цикл данного метода характеризуется значительным числом итераций.

Основными этапами метода COMET являются:

1. Этап моделирования функциональных требований (Requirements Modeling), которое выполняется в терминах акторов и прецедентов. На данном этапе основное внимание уделяется сбору и классификации требований к системе, в то время как сама система рассматривается как черный ящик.
2. Этап аналитического моделирования (Analysis Modeling), которое выполняется в терминах сущностной модели. На данном этапе основное внимание уделяется предметной области, при этом структура сущностной модели описывается с помощью статического представления модели, а характер поведения — с помощью динамического представления модели. Статическое представление модели выполняется в терминах классов (объектов) предметной области и отношений между ними, в то время как динамическое представление модели выполняется в терминах взаимодействия между объектами.
3. Этап архитектурного (имитационного) моделирования (Design Modeling), которое выполняется в терминах структуры имитационной модели (классы и отношения между ними). На данном этапе основное внимание уделяется объектной и временной декомпозиции сущностной модели, формулируются базовые критерии разбиения системы на составные части (подсистемы, модули и проч.). Статическое представление структуры имитационной модели приводится на диаграммах кооперации и классов. Динамическое представление приводится на диаграммах состояний и последовательности.
4. Этап программного моделирования (Program Modeling), которое выполняется в терминах программной модели (атрибуты и операции классов). На данном этапе особое внимание уделяется программной реализации имитационной модели. В частности, статическое представление имитационной модели детализируется до атрибутов и операций классов, а также до законченных иерархий и других отношений между классами. Динамическое представление модели детализируется до полного описания активных составляющих, какими являются задачи (thread), а также проектируются интерфейсы для обмена сообщениями и рассматриваются синхнонные, асинхронные, групповые и брокерские коммуникации. В случае необходимости, например, при проектировании системы реального времени, производится анализ производительности

новой системы на основе метода монотонного анализа частот. Кроме этого, на данном этапе происходит транслирование описания системы из нотаций графического и текстового описания моделей в нотацию конкретного машино-интерпретируемого языка, который принято называть языком программирования.

Ключевые вопросы:

1. Назначение и концепции метода, задействованные роли, жизненный цикл разработки, понятия объекта, класса, операции, задачи;
2. Фазы проектирования информационной модели и программной системы; связь между фазами;
3. Определение понятий «прецедент», «ассоциация», «сущность», «событие», «сообщение», «контекст модели», «контекст системы», «архитектурный стиль».

Литература:

1 Избачков Ю.С. Информационные системы: учеб.:рек. Мин. обр. и науки РФ/ Ю. С. Избачков, В. Н. Петров. -2-е изд. -СПб.: Питер, 2008.-656 с.
2 Гвоздева Т.В. Проектирование информационных систем: учеб. пособие: рек. УМО/ Т.В. Гвоздева, Б.А. Баллод. -Ростов н/Д: Феникс, 2009.-509 с.
3 Хомоненко А. Д. Базы данных: учеб.: рек. УМО / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев; под ред. А. Д. Хомоненко. -6-е изд., доп. -СПб.: КОРОНА-Век, 2009.-736 с.

Практическое занятие 3. Проектирование информационной модели.

Информационная модель- это совокупность диаграмм типа "сущность - связь". Модели "сущность - связь" определяют структуру и взаимные связи используемой в системе информации, в полной мере отображающей её работу.

Цель информационной модели заключается в выработке непротиворечивой интерпретации данных и взаимосвязей между ними, что необходимо для интеграции, совместного использования и управления целостностью данных.

Информационную модель мы выполняем в стандарте IDEF1X.

Стандарт IDEF1X был разработан в 1983 году в рамках проекта военного ведомства США "Интегрированные системы информационной поддержки" (ICAM) как методология семантического моделирования данных. Она стала расширением методологии IDEF1 и позволила логически и физически объединять в сеть неоднородные вычислительные системы.

Методология IDEF1X - один из подходов к семантическому моделированию данных, основанный на концепции Сущность-Связь. Это инструмент для анализа информационной структуры систем различной природы. Информационная модель, построенная с помощью IDEFlX- методологии, представляет логическую структуру информации об объектах системы. Эта информация является необходимым дополнением функциональной IDEF0- модели поскольку детализирует объекты, которыми манипулируют функции системы. Концептуально IDEF1X- модель можно рассматривать как проект логической схемы базы данных для проектируемой системы.

Информационные модели типа "сущность - связь" являются логическими моделями и не зависят от реализации баз данных и методов доступа.

Пример информационной модели представлен на рисунке.

Основными компонентами IDEF1X- модели являются:

- Сущности, представляющие множество реальных или абстрактных предметов (людей, объектов, мест, событий, состояний, идей, пар предметов и т.д). Они изображаются блоками. Могут быть:
  - независимые от идентификатора сущности;
  - зависимые от идентификатора сущности.
- Отношения между этими предметами, изображаемые соединяющими блоки линиями. Могут быть:
  - отношения, идентифицирующие связи;
  - отношения, не идентифицирующие связи;

    - отношения категоризации;
    - неспецифические отношения.
- Характеристики сущностей, изображаемые именами атрибутов внутри блоков. Атрибуты могут быть:
    - неключевыми;
    - первичными ключами;
    - альтернативными ключами;
    - внешними ключами.

Сущность – это совокупность похожих объектов/экземпляров (людей, мест, предметов, событий), которая именуется общим существительным, обладает ключом (однозначно идентифицирующим каждый экземпляр) и имеет один или несколько атрибутов (описывающих каждый экземпляр).

Каждая сущность может обладать любым количеством отношений с другими сущностями. Сущность является независимой, если каждый экземпляр сущности может быть однозначно идентифицирован без определения его отношений с другими сущностями. Независимая сущность на диаграмме изображается блоком с прямыми углами. Сущность называется зависимой, если однозначная идентификация экземпляра сущности зависит от его отношения к другой сущности. Если сущность зависима от идентификатора, то углы блока закругляются.

Например, сущности Клиент, Издание являются независимыми, а сущность Заказ клиента - *зависимой.*

Отношение связи, называемое также отношением родитель-потомок, - это связь между сущностями, при которой каждый экземпляр сущности-родитель ассоциирован с произвольным (в том числе нулевым) количеством экземпляров сущности-потомком, а - каждый экземпляр сущности-потомка ассоциирован в точности с одним экземпляром сущности-родителя.

Идентифицирующим отношением называется отношение, при котором экземпляр сущности-потомка однозначно определяется своей связью с сущностью-родителем. В противном случае отношение называется неидентифицирующим. Например, специфическое отношение связи будет существовать между сущностью Клиент и Заказ клиента, если Клиент имеет ноль, один или более Заказов, а каждый Заказ принадлежит только одному Клиенту. Отношение связи изображается линией, проводимой между сущностью-родителем и сущностью-потомком с точкой на конце линии у сущности-потомка.

Идентифицирующее отношение изображается сплошной линией, пунктирная линия изображает неидентифицирующее отношение.

Отношению дается имя, выражаемое грамматическим оборотом глагола. Имя отношения всегда формируется с точки зрения родителя, так что может быть образовано предложение, если соединить имя сущности-родителя, имя отношения, выражение мощности и имя сущности-потомка.

Мощность определяет какое количество экземпляров сущности-потомка может существовать для каждого экземпляра сущности-родителя.

В IDEF1X могут быть выражены следующие мощности отношений:

- Каждый экземпляр сущности-родителя может иметь ноль, один или более связанных с ним экземпляров сущности-потомка (эта мощность устанавливается по умолчанию);
- Каждый экземпляр сущности-родителя должен иметь не менее одного связанного с ним экземпляра сущности-потомка (буква P (positive), помещенная рядом с точкой);
- Каждый экземпляр сущности-родителя может иметь не более одного связанного с ним экземпляра сущности-потомка (буква Z (zero), помещенная рядом с точкой);
- Каждый экземпляр сущности-родителя связан с некоторым фиксированным числом экземпляров сущности-потомка. Если мощность в точности равна некоторому числу N, это число (целое, положительное) помещается около точки.

Атрибут - это характеристика или элемент данных, описывающий что-либо в сущности. Каждому атрибуту присваивается уникальное имя, обозначающее его смысл (например,

цвет волос) и значение (например, коричневый). Одно и то же значение не может соответствовать различным именам.

Сущность может обладать любым количеством атрибутов. Атрибуты изображаются в виде списка их имен внутри блока ассоциированной сущности, причем каждый атрибут занимает отдельную строку. Сущность должна обладать атрибутом или комбинацией атрибутов, чьи значения однозначно определяют каждый экземпляр сущности. Эти атрибуты образуют первичный ключ сущности.

Например, атрибут "ИдЗаказа" однозначно идентифицирует экземпляр сущности Заказ клиента, поэтому он может быть атрибутом первичного ключа.

Такие атрибуты размещаются наверху списка и отделяются от других атрибутов горизонтальной чертой. Другие атрибуты сущности являются неключевыми.

Ключевые вопросы:

1. Моделирование прецедентов, их выявление и документирование;
2. Статическое моделирование: ассоциации между классами; иерархии композиции/агрегирования и обобщения/специализации;
3. Выявление и документирование сущностных классов; критерии выявления; контекст модели.

Литература:

1 Избачков Ю.С. Информационные системы: учеб.:рек. Мин. обр. и науки РФ/ Ю. С. Избачков, В. Н. Петров. -2-е изд. -СПб.: Питер, 2008.-656 с.
2 Гвоздева Т.В. Проектирование информационных систем: учеб. пособие: рек. УМО/ Т.В. Гвоздева, Б.А. Баллод. -Ростов н/Д: Феникс, 2009.-509 с.
3 Хомоненко А. Д. Базы данных: учеб.: рек. УМО / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев; под ред. А. Д. Хомоненко. -6-е изд., доп. -СПб.: КОРОНА-Век, 2009.-736 с.

Практическое занятие 4. Проектирование информационной системы.

Если вы хотите создать качественную модель, то придется прибегать к помощи аналитиков, хорошо владеющих CASE-технологией. Однако это не означает, что построением и контролем информационной модели должны заниматься только аналитики. Помощь коллег также может оказаться весьма полезной. Привлекайте их к проверке поставленной цели и к детальному изучению построенной модели как с точки зрения логики, так и с точки зрения учета аспектов предметной области. Большинство людей легче находят недостатки в чужой работе.

Регулярно представляйте вашу информационную модель или ее отдельные фрагменты, относительно которых у вас возникают сомнения, на одобрение пользователей. Особое внимание уделяйте исключениям из правил и ограничениям.

Основной гарантией качества сущности является ответ на вопрос, действительно ли объект является сущностью, то есть важным объектом или явлением, информация о котором должна храниться в базе данных.

Список проверочных вопросов для сущности:

- Отражает ли имя сущности суть данного объекта?
- Нет ли пересечения с другими сущностями?
- Имеются ли хотя бы два атрибута?
- Всего атрибутов не более восьми?
- Есть ли синонимы/омонимы данной сущности?
- Сущность определена полностью?
- Есть ли уникальный идентификатор?
- Имеется ли хотя бы одна связь?
- Существует ли хотя бы одна функция по созданию, поиску, корректировке, удалению, архивированию и использованию значения сущности?
- Ведется ли история изменений?
- Имеет ли место соответствие принципам нормализации данных?

- Нет ли такой же сущности в другой прикладной системе, возможно, под другим именем?
- Не имеет ли сущность слишком общий смысл?
- Достаточен ли уровень обобщения, воплощенный в ней?

Список проверочных вопросов для подтипа:

- Отсутствуют ли пересечения с другими подтипами?
- Имеет ли подтип какие-нибудь атрибуты и/или связи?
- Имеют ли они все свои собственные уникальные идентификаторы или наследуют один на всех от супертипа?
- Имеется ли исчерпывающий набор подтипов?
- Не является ли подтип примером вхождения сущности?
- Знаете ли вы какие-нибудь атрибуты, связи и условия, отличающие данный подтип от других?

Следует выяснить, а действительно ли это атрибуты, то есть описывают ли они тем или иным образом данную сущность.

Список проверочных вопросов для атрибута:

- Является ли наименование атрибута существительным единственного числа, отражающим суть обозначаемого атрибутом свойства?
- Не включает ли в себя наименование атрибута имя сущности (этого быть не должно)?
- Имеет ли атрибут только одно значение в каждый момент времени?
- Отсутствуют ли повторяющиеся значения (или группы)?
- Описаны ли формат, длина, допустимые значения, алгоритм получения и т.п.?
- Не может ли этот атрибут быть пропущенной сущностью, которая пригодилась бы для другой прикладной системы (уже существующей или предполагаемой)?
- Не может ли он быть пропущенной связью?
- Нет ли где-нибудь ссылки на атрибут как на «особенность проекта», которая при переходе на прикладной уровень должна исчезнуть?
- Есть ли необходимость в истории изменений?
- Зависит ли его значение только от данной сущности?
- Если значение атрибута является обязательным, всегда ли оно известно?
- Есть ли необходимость в создании домена для этого и ему подобных атрибутов?
- Зависит ли его значение только от какой-то части уникального идентификатора?
- Зависит ли его значение от значений некоторых атрибутов, не включенных в уникальный идентификатор?

Нужно выяснить, отражают ли связи действительно важные отношения, наблюдаемые между сущностями.

Список проверочных вопросов для связи:

- Имеется ли ее описание для каждой участвующей стороны, точно ли оно отражает содержание связи и вписывается ли в принятый синтаксис?
- Участвуют ли в ней только две стороны?

Не является ли связь переносимой?

- Заданы ли степень связи и обязательность для каждой стороны?
- Допустима ли конструкция связи?

Не относится ли конструкция связи к редко используемым?

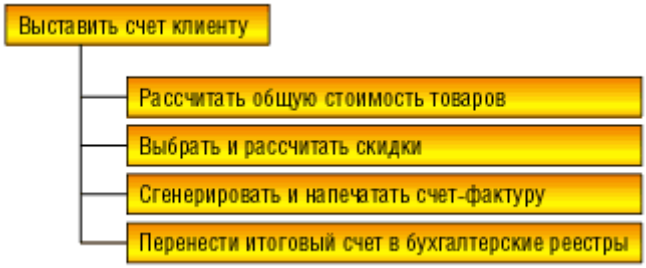
- Не является ли она избыточной?
- Не изменяется ли она с течением времени?
- Если связь обязательная, всегда ли она отражает отношение к сущности, представляющей противоположную сторону?

Для исключающей связи:

- Все ли концы связей, покрываемые исключающей дугой, имеют один и тот же тип обязательности?
- Все ли из них относятся к одной и той же сущности?

- Обычно дуги пересекают разветвляющиеся концы - что вы можете сказать о данном случае?
- Связь может покрываться только одной дугой. Так ли это?
- Все ли концы связей, покрываемые дугой, входят в уникальный идентификатор?

Часто аналитикам приходится описывать достаточно сложные бизнес-процессы. В этом случае прибегают к функциональной декомпозиции, которая показывает разбиение одного процесса на ряд более мелких функций до тех пор, пока каждую из них уже нельзя будет разбить без ущерба для смысла. Конечный продукт декомпозиции представляет собой иерархию функций, на самом нижнем уровне которой находятся атомарные с точки зрения смысловой нагрузки функции. Приведем простой пример такой декомпозиции. Рассмотрим простейшую задачу выписки счета клиенту при отпуске товара со склада при условии, что набор товаров, которые хочет приобрести клиент, уже известен (не будем рассматривать в данном примере задачу выбора товаров).



Пример декомпозиции

Очевидно, что операция выбора и расчета скидок может быть также разбита на более мелкие операции, например на расчет скидок за приверженность (клиент покупает товары в течение долгого времени) и на расчет скидок за количество покупаемого товара. Атомарные функции описываются подробно, например с помощью DFD и STD. Очевидно, что такое описание функций не исключает и дополнительное словесное описание (например, комментарии).

Следует отметить, что на этапе анализа следует уделить внимание функциям анализа и обработки возможных ошибок и отклонений от предполагаемого эталона работы системы. Следует выделить наиболее критичные для работы системы процессы и обеспечить для них особенно строгий анализ ошибок. Обработка ошибок СУБД (коды возврата), как правило, представляет собой обособленный набор функций или одну-единственную функцию.

На этапе анализа происходит уточнение выбранных для конечной реализации аппаратных и программных средств. Для этого могут привлекаться группы тестирования, технические специалисты. При проектировании информационной системы важно учесть и дальнейшее развитие системы, например рост объемов обрабатываемых данных, увеличение интенсивности потока запросов, изменение требований надежности информационной системы.

На этапе анализа определяются наборы моделей задач для получения сравнительных характеристик тех или иных СУБД, которые рассматривались на этапе определения стратегии для реализации информационной системы. На этапе определения стратегии может быть осуществлен выбор одной СУБД. Данных о системе на этапе анализа уже намного больше, и они более подробны. Полученные данные, а также характеристики, переданные группами тестирования, могут показать, что выбор СУБД на этапе определения стратегии был неверным и что выбранная СУБД не может удовлетворять тем или иным требованиям информационной системы. Такие же данные могут быть получены относительно выбора аппаратной платформы и операционной системы. Получение подобных результатов инициирует изменение данных, полученных на этапе определения стратегии, например пересчитывается смета затрат на проект.

Выбор средств разработки также уточняется на этапе анализа. В силу того что этап анализа дает более полное представление об информационной системе, чем оно было на эта-

пе определения стратегии, план работ может быть скорректирован. Если выбранное на предыдущем этапе средство разработки не позволяет выполнить ту или иную часть работ в заданный срок, то принимается решение об изменении сроков (как правило, это увеличение срока разработки) или о смене средства разработки. Осуществляя выбор тех или иных средств, следует учитывать наличие высококвалифицированного персонала, который владеет выбранными средствами разработки, а также наличие администраторов выбранной СУБД. Эти рекомендации также будут уточнять данные этапа выбора стратегии (совокупность условий, при которых предполагается эксплуатировать будущую систему).

Уточняются также ограничения, риски, критические факторы. Если какие-либо требования не могут быть удовлетворены в информационной системе, реализованной с использованием СУБД и программных средств, выбранных на этапе определения стратегии, то это также инициирует уточнение и изменение получаемых данных (в конечном итоге сметы затрат и планов работ, а возможно, и изменение требований заказчика к системе, например их ослабление). Более подробно описываются те возможности, которые не будут реализованы в системе.

Ключевые вопросы:

1. Разбиение на классы и объекты;
2. Критерии разбиения, категории классов, подсистемы;
3. Конечные автоматы и диаграммы состояний: события, состояния, условия переходов, плоские, иерархические и параллельные диаграммы состояний;
4. Динамическое моделирование: взаимодействие объектов, диаграммы взаимодействия, динамический анализ;
5. Проектирование архитектуры системы: существующие архитектурные стили, декомпозиция, критерии декомпозиции, разделение обязанностей, консолидированные диаграммы кооперации;
6. Проектирование архитектуры распределенных систем: конфигурируемые архитектуры, декомпозиция, критерии декомпозиции, интерфейсы подсистем, управление транзакциями, распределение данных;
7. Разбиение на задачи: критерии разбиения, инверсия и архитектура задач, межзадачные коммуникации, спецификации поведения задач;
8. Проектирование классов: классы, скрывающие данные, алгоритмы, бизнес-логику, интерфейс пользователя; применение наследования в проектировании классов; спецификации интерфейсов классов;
9. Детальное проектирование: составные задачи, синхронизация доступа, «разъемы» для межзадачных коммуникаций, логика упорядочения событий;
10. Анализ производительности: теория планирования в реальном времени, анализ с помощью последовательности событий, смешанный анализ; пересмотр проекта; оценка и изменение параметров производительности.

Литература:

1 Избачков Ю.С. Информационные системы: учеб.:рек. Мин. обр. и науки РФ/ Ю. С. Избачков, В. Н. Петров. -2-е изд. -СПб.: Питер, 2008.-656 с.
2 Гвоздева Т.В. Проектирование информационных систем: учеб. пособие: рек. УМО/ Т.В. Гвоздева, Б.А. Баллод. -Ростов н/Д: Феникс, 2009.-509 с.
3 Хомоненко А. Д. Базы данных: учеб.: рек. УМО / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев; под ред. А. Д. Хомоненко. -6-е изд., доп. -СПб.: КОРОНА-Век, 2009.-736 с.

Практическое занятие 5. Тестирование информационной системы.

Тестирование — неотъемлемая часть жизненного цикла любого программного продукта. В финансовых организациях, бизнес которых существенно зависит от возможностей IT-инфраструктуры, тестирование является важным подготовительным этапом к внедрению новых информационных систем.

Функциональное тестирование

- *Компонентное*: проверка функциональности на уровне отдельных компонент информационной системы.
- *Интеграционное*: проверка функциональности при интеграции нескольких компонент.
- *Системное*: проверка функциональности всей системы.
- *Приемочное*: подтверждение функциональности системы на основе контрольных примеров.
- *Регрессионное*: проверка ранее работавшей функциональности в случае внесения изменений.
- *Автоматизация регрессионного тестирования*: разработка автоматизированных скриптов, выполняющих тестовые операции по проверке фактического результата через пользовательский или программный интерфейс
- *Совместимости*: проверка функциональности при использовании разных версий предположительно совместимого программного обеспечения.
- *Исследовательское*: быстрое выявление наиболее критичных дефектов функциональности без длительной подготовки и обеспечения тестового покрытия.

Нефункциональное тестирование

- *Производительности*: исследование характеристик производительности системы для заданного уровня пользовательской нагрузки, объема данных и программно-аппаратной конфигурации.
- *Нагрузочное*: исследование характеристик производительности системы для разных уровней пользовательской нагрузки.
- *Объемов*: исследование характеристик производительности системы на разных объемах данных.
- *Конфигурационное*: исследование характеристик производительности системы на разных программно-аппаратных конфигурациях.
- *Анализ и оптимизация производительности*: выявление узких мест производительности системы и формирование предложений по их устранению.
- *Надежности*: исследование отказов системы при относительно длительной эксплуатации в условиях номинальной или пиковой нагрузки.
- *Стрессовое*: исследование поведения системы в условиях недостатка аппаратных или программных ресурсов.
- *Отказоустойчивости*: исследование возможностей восстановления нормальной работы системы после аппаратных и программных сбоев.
- *Инсталляционное*: проверка комплектности дистрибутивов и их корректной установки в соответствии с поставляемой инструкцией.
- *Документации:* проверка соответствия документации заданным критериям качества.

Функциональное тестирование обеспечивает проверку корректного выполнения операций, автоматизированных информационной системой. В каждом проекте при выборе методов тестирования производится обследование возможностей тестирования и учет требований заказчика по срокам, бюджету и необходимому уровню качества.

Функциональное тестирование может проводиться с разной степенью тестового покрытия:

- тестирование по актуализированным спецификациям (тестовым требованиям)
- тестирование по существующим спецификациям
- поиск дефектов на основе знаний аналогичных систем
- проверка на основе контрольных примеров.

Для достижения максимального эффекта применяются разные способы построения тестов:

- пошаговое описание тестов
- чек-листы
- моделирование тестируемой системы

- сравнение системы с предыдущей версией
- автоматизированные GUI-тесты
- модульные (unit) тесты
- тесты, управляемые данными
- автоматическая генерация тестовых сценариев.

Результаты тестирования оформляются в отчете о тестировании, который содержит:

- информацию о версии протестированной системы
- фактические условия и ограничения выполненного тестирования
- перечень проверенных функциональных возможностей системы
- данные о количестве проведенных тестов
- данные о количестве обнаруженных дефектов и их список
- распределение дефектов по приоритетам и состояниям
- заключение о качестве информационной системы.

Высокое качество функционального тестирования обеспечивают:

- актуализация требований к системе: на этапе подготовки к тестированию производится анализ требований к системе, выявление и устранение обнаруженных недостатков
- необходимые широта и глубина тестового покрытия: в плане тестирования указываются функциональные возможности системы, которые будут протестированы, и стратегии формирования тестов, которые будут для этого задействованы — разбиение данных на классы эквивалентности, контроль специфичных значений параметров, контроль обхода ветвей алгоритмов, проверка сложных условий и т. д.
- качественное описание зарегистрированных дефектов: дефекты корректно описывают недостатки системы и содержат достаточно информации для их дальнейшей локализации.

Нефункциональное тестирование

Наряду с функциональными характеристиками программного обеспечения высокую важность имеют такие его характеристики, как производительность, надежность и отказоустойчивость. На исследование данных показателей направлено нефункциональное тестирование.

Почему важно исследовать характеристики производительности, надежности и отказоустойчивости систем?

Эти характеристики влияют на время выполнения бизнес-процессов, допустимые объемы бизнес-операций и затраты на оборудование. Нередко высокая пользовательская нагрузка приводит к сбоям в информационных системах. В этих случаях характеристики производительности, надежности и отказоустойчивости приобретают критический характер — отказ информационной системы может вызвать приостановку операционной деятельности организации, что приводит к серьезным финансовым и репутационным потерям.

Подготовка к тестированию, включает:

- анализ специфики бизнеса и выявление необходимых видов тестирования
- выявление нагрузочных профилей для разных режимов и периодов использования системы
- определение состава и конфигурации тестового стенда для корректного моделирования поведения исследуемой системы
- определение состава журналируемых характеристик системы
- разработку автоматизированных нагрузочных скриптов и эмуляторов.

Результаты тестирования оформляются в отчете, который содержит:

- информацию о версии и конфигурации протестированной системы
- фактические условия и ограничения выполненного тестирования
- характеристики производительности, надежности и отказоустойчивости системы
- описание узких мест системы
- рекомендации по оптимизации системы
- заключение о качестве системы.

Преимущества аутсорсинга тестирования

Аутсорсинг тестирования заключается в привлечении к тестированию проектной команды специалистов, предоставленной внешней компанией. В состав команды по тестированию, входят опытный руководитель проекта, ведущий инженер по обеспечению качества программного обеспечения и квалифицированные тестировщики. В таком составе команда способна выполнить проект любой сложности и обеспечить все преимущества аутсорсинга тестирования:

- доступность персонала: к моменту начала работ по тестированию команда уже сформирована и готова к выполнению работ
- предсказуемость результата: контрактная форма выполнения работ обязывает к четкому определению результатов
- технологичность: для достижения предсказуемости результата компания, выполняющая аутсорсинг тестирования, использует специальные технологии. Это упорядочивает все промежуточные этапы тестирования и является необходимым условием выполнения работ с высоким качеством
- объективность результатов: результаты тестирования заслуживают большего доверия, так как независимая команда не заинтересована в сокрытии обнаруженных проблем в информационной системе
- высокая ответственность: уровень ответственности, принимаемый на себя внешней компанией, на порядок выше, чем уровень ответственности отдельного сотрудника собственного отдела тестирования. Это приводит к значительному улучшению результатов работ по всем показателям.

Ключевые вопросы:

1. Типы тестирования;
2. Место тестирования в жизненном цикле разработки ИС;
3. Типовые роли, участвующие в процессе тестирования;

Литература:

1 Избачков Ю.С. Информационные системы: учеб.:рек. Мин. обр. и науки РФ/ Ю. С. Избачков, В. Н. Петров. -2-е изд. -СПб.: Питер, 2008.-656 с.

2 Гвоздева Т.В. Проектирование информационных систем: учеб. пособие: рек. УМО/ Т.В. Гвоздева, Б.А. Баллод. -Ростов н/Д: Феникс, 2009.-509 с.

3 Хомоненко А. Д. Базы данных: учеб.: рек. УМО / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев; под ред. А. Д. Хомоненко. -6-е изд., доп. -СПб.: КОРОНА-Век, 2009.-736 с.

Практическое занятие 6. Системное программное обеспечение.

Программное обеспечение

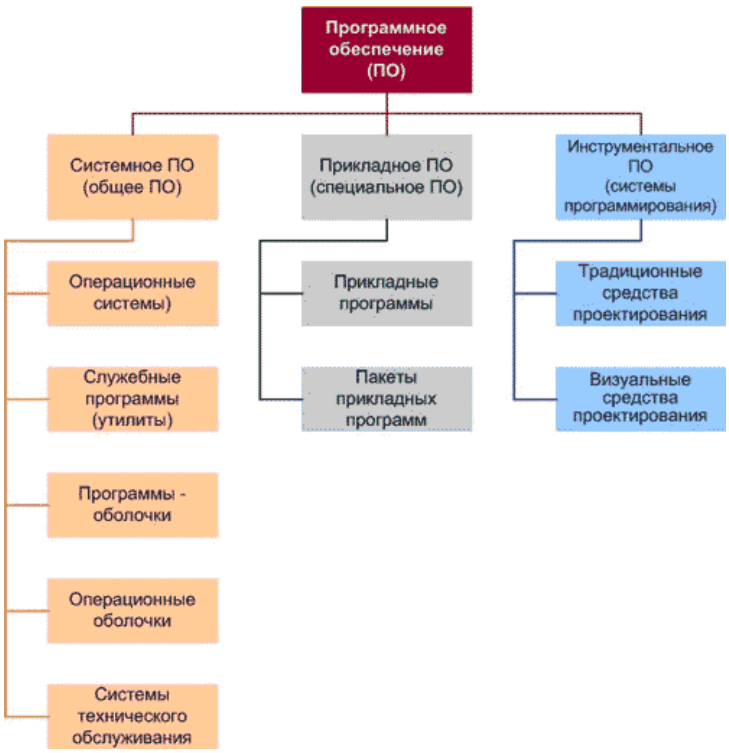
Совокупность программ, предназначенная для решения задач на ПК, называется программным обеспечением. Состав программного обеспечения ПК называют программной конфигурацией.

Программное обеспечение, можно условно разделить на три категории:

системное ПО (программы общего пользования), выполняющие различные вспомогательные функции, например создание копий используемой информации, выдачу справочной информации о компьютере, проверку работоспособности устройств компьютера и т.д.

прикладное ПО, обеспечивающее выполнение необходимых работ на ПК: редактирование текстовых документов, создание рисунков или картинок, обработка информационных массивов и т.д.

инструментальное ПО (системы программирования), обеспечивающее разработку новых программ для компьютера на языке программирования.



Системное ПО

Это программы общего пользования не связаны с конкретным применением ПК и выполняют традиционные функции: планирование и управление задачами, управления вводом-выводом и т.д.

Другими словами, системные программы выполняют различные вспомогательные функции, например, создание копий используемой информации, выдачу справочной информации о компьютере, проверку работоспособности устройств компьютера и т.п.

К системному ПО относятся:

операционные системы (эта программа загружается в ОЗУ при включении компьютера)

программы – оболочки (обеспечивают более удобный и наглядный способ общения с компьютером, чем с помощью командной строки DOS, например, Norton Commander)

операционные оболочки – интерфейсные системы, которые используются для создания графических интерфейсов, мультипрограммирования и.т.

Драйверы (программы, предназначенные для управления портами периферийных устройств, обычно загружаются в оперативную память при запуске компьютера)

утилиты (вспомогательные или служебные программы, которые представляют пользователю ряд дополнительных услуг)

К утилитам относятся:

диспетчеры файлов или файловые менеджеры

средства динамического сжатия данных (позволяют увеличить количество информации на диске за счет ее динамического сжатия)

средства просмотра и воспроизведения

средства диагностики; средства контроля позволяют проверить конфигурацию компьютера и проверить работоспособность устройств компьютера, прежде всего жестких дисков

средства коммуникаций (коммуникационные программы) предназначены для организации обмена информацией между компьютерами

средства обеспечения компьютерной безопасности (резервное копирование, антивирусное ПО).

Необходимо отметить, что часть утилит входит в состав операционной системы, а другая часть функционирует автономно. Большая часть общего (системного) ПО входит в состав ОС. Часть общего ПО входит в состав самого компьютера (часть программ ОС и контроли-

рующих тестов записана в ПЗУ или ППЗУ, установленных на системной плате). Часть общего ПО относится к автономными программам и поставляется отдельно.

Ключевые вопросы:

1. Существующие типы системного ПО;
2. Место системного ПО в жизненном цикле предприятия;
3. Операционные системы: назначение, архитектура, функции;
4. Системы управления базами данных (СУБД): назначение, архитектура, функции;
5. Программное обеспечение «среднего» слоя: назначение, архитектура, функции;
6. Эксплуатация системного ПО; задачи персонала по эксплуатации.

Литература:

1 Избачков Ю.С. Информационные системы: учеб.:рек. Мин. обр. и науки РФ/ Ю. С. Избачков, В. Н. Петров. -2-е изд. -СПб.: Питер, 2008.-656 с.
2 Гвоздева Т.В. Проектирование информационных систем: учеб. пособие: рек. УМО/ Т.В. Гвоздева, Б.А. Баллод. -Ростов н/Д: Феникс, 2009.-509 с.
3 Хомоненко А. Д. Базы данных: учеб.: рек. УМО / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев; под ред. А. Д. Хомоненко. -6-е изд., доп. -СПб.: КОРОНА-Век, 2009.-736 с.

Практическое занятие 7. Прикладное программное обеспечение.

Прикладные программы могут использоваться автономно или в составе программных комплексов или пакетов. Прикладное ПО – программы, непосредственно обеспечивающие выполнение необходимых работ на ПК: редактирование текстовых документов, создание рисунков или картинок, создание электронных таблиц и т.д.

Пакеты прикладных программ – это система программ, которые по сфере применения делятся на проблемно – ориентированные, пакеты общего назначения и интегрированные пакеты. Современные интегрированные пакеты содержат до пяти функциональных компонентов: тестовый и табличный процессор, СУБД, графический редактор, телекоммуникационные средства.

К прикладному ПО, например, относятся:

Комплект офисных приложений MS OFFICE

Бухгалтерские системы

Финансовые аналитические системы

Интегрированные пакеты делопроизводства

CAD – системы (системы автоматизированного проектирования)

Редакторы HTML или Web – редакторы

Браузеры – средства просмотра Web – страниц

Графические редакторы

Экспертныесистемы

И так далее.

Инструментальное ПО

Инструментальное ПО или системы программирования - это системы для автоматизации разработки новых программ на языке программирования.

В самом общем случае для создания программы на выбранном языке программирования (языке системного программирования) нужно иметь следующие компоненты:

1. Текстовый редактор для создания файла с исходным текстом программы.

2. Компилятор или интерпретатор. Исходный текст с помощью программы-компилятора переводится в промежуточный объектный код. Исходный текст большой программы состоит из нескольких модулей (файлов с исходными текстами). Каждый модуль компилируется в отдельный файл с объектным кодом, файлы надо объединить в одно целое.

3. Редактор связей или сборщик, который выполняет связывание объектных модулей и формирует на выходе работоспособное приложение – исполнимый код.

Исполнимый код – это законченная программа, которую можно запустить на любом компьютере, где установлена операционная система, для которой эта программа создавалась.

Как правило, итоговый файл имеет расширение .EXE или .COM.

В последнее время получили распространение визуальный методы программирования (с помощью языков описания сценариев), ориентированные на создание Windows-приложений. Этот процесс автоматизирован в средах быстрого проектирования. При этом используются готовые визуальные компоненты, которые настраиваются с помощью специальных редакторов.

Наиболее популярные редакторы (системы программирования программ с использованием визуальных средств) визуального проектирования:

Borland Delphi - предназначен для решения практически любых задачи прикладного программирования

Borland C++ Builder – это отличное средство для разработки DOS и Windows приложений

Microsoft Visual Basic – это популярный инструмент для создания Windows-программ

Microsoft Visual C++ - это средство позволяет разрабатывать любые приложения, выполняющиеся в среде ОС типа Microsoft Windows

Ключевые вопросы:

1. Существующие типы прикладного ПО.2. Место системного ПО в жизненном цикле предприятия;
3. Корпоративные информационные системы: назначение, архитектура, функции;
4. Системы управления базами данных (СУБД): назначение, архитектура, функции;
5. Программное обеспечение «среднего» слоя: назначение, архитектура, функции;
6. Эксплуатация прикладного ПО; задачи персонала по эксплуатации.

Литература:

1 Избачков Ю.С. Информационные системы: учеб.:рек. Мин. обр. и науки РФ/ Ю. С. Избачков, В. Н. Петров. -2-е изд. -СПб.: Питер, 2008.-656 с.

2 Гвоздева Т.В. Проектирование информационных систем: учеб. пособие: рек. УМО/ Т.В. Гвоздева, Б.А. Баллод. -Ростов н/Д: Феникс, 2009.-509 с.

3 Хомоненко А. Д. Базы данных: учеб.: рек. УМО / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев; под ред. А. Д. Хомоненко. -6-е изд., доп. -СПб.: КОРОНА-Век, 2009.-736 с.

Практическое занятие 8. Интеграция программного обеспечения.

Интеграция корпоративных приложений в единое информационное пространство дает организации следующие преимущества:

- «Бесшовное» объединение различных систем на базе универсального формата обмена данными, такого как XML
- Увеличение эффективности работы информационных подсистем заказчика
- Снижение общей стоимости владения инфосистемой
- Снижение стоимости разработки и поддержки процессов интеграции
- Уменьшение затрат на объединение ERP-системы (такой, как SAP, Oracle EBS, Microsoft Dynamics и т.д.) с другим программным обеспечением (таким, как системы электронного документооборота) в единую информационную систему
- Сокращение расходов на разработку интеграционных компонент (адаптеры, конверторы).

В последнее время разработчики встроенных систем управления все чаще и чаще сталкиваются с задачами перевода прикладного программного обеспечения (ПО) встроенных систем реального времени с одной программно-аппаратной платформы на другую. Как правило, наследованный код прикладного ПО работает под управлением операционной системы реального времени (ОСРВ). ОСРВ осуществляет диспетчирование задач реального времени, обработку прерываний от внешних устройств, предоставляет набор примитивов для организации межзадачного взаимодействия и другие системные средства. Конкретный набор средств, предоставляемых ОСРВ, определятся архитектурой ОСРВ, областью применения, аппаратной платформой, на которой работает ОСРВ, а также требованиями заказчиков, вы-

сказанными на этапе проектирования ОСРВ. В результате различие между коммерческими системами, представленными на рынке ОСРВ, достаточно велико.

Простейшие системы (например OSEK или RTXC) работают на 8-, 16- и 32-разрядных микроконтроллерах и предоставляют только базовые средства по диспетчеризации задач и обработки прерываний. Как правило, подобные системы (назовем их базовыми ОСРВ) не поддерживают стандартных интерфейсов разработки прикладного ПО (например POSIX) и не предоставляют механизмов защиты памяти. Достоинствами базовых ОСРВ является компактность систем, открытость исходного кода систем, прозрачность внутренней архитектуры и обеспечение высокого и гарантированного времени реакции системы на внешние воздействия.

Недостатками подобных систем является их низкая масштабируемость (как правило, отладка системы существенно усложняется, если число одновременно запущенных задач превышает несколько десятков), сложность интеграции компонентов системы в единое целое (отсутствие защиты памяти), сложность повторного использования программных компонентов, изначально разработанных для других проектов (отсутствие поддержки стандартных интерфейсов разработки прикладного ПО), ограниченность средств отладки (как правило, отладка кода прикладного ПО может быть завершена только на целевой аппаратной платформе) и некоторые другие ограничения, обсуждение которых не является предметом данной статьи.

Более сложные системы (например VxWorks, QNX и Linux) работают, как правило, на 32- и 64-разрядных микроконтроллерах и предоставляют не только базовые средства по диспетчеризации задач и обработки прерываний, но и широкий набор дополнительных средств по организации межпотокового взаимодействия, по локализации сбоев в прикладном ПО и т.д. Как правило, подобные системы (назовем их расширенными ОСРВ) поддерживают не только внутренние интерфейсы разработки прикладного ПО, но и ряд стандартных интерфейсов, включая полную либо частичную поддержку POSIX стандарта. Достоинствами расширенных ОСРВ является поддержка системы защиты памяти, что повышает надежность конечной системы и снижает трудозатраты на этапе интеграции компонентов системы в единое целое, поддержка стандартных интерфейсов создания прикладного ПО (появляется возможность интеграции программных компонентов, изначально написанных для других продуктов), хорошая масштабируемость систем и чрезвычайно богатые средства отладки прикладного ПО (как правило, прикладное ПО может быть отлажено на инструментальной машине до того, как загружено на целевую аппаратную платформу). К недостаткам подобного рода систем необходимо отметить повышенные требования, предъявляемые к ресурсам целевой аппаратной платформы и увеличение времени реакции системы на внешние воздействия (не всегда возможно определить граничное значение времени реакции).

В некоторых сегментах рынка для 32-раз рядных микроконтроллеров видна тенденция к переходу от базовых ОСРВ к расширенным. Рассмотрим один из подходов к решению задачи перевода прикладного ПО встроенных систем реального времени с базовой ОСРВ на расширенную с сохранением существующих функций продукта, реализованных с использованием базовой ОСРВ.

Нашей задачей является переход с базовой ОСРВ на расширенную с сохранением функций встроенной системы управления, ранее реализованных на основе базовой ОСРВ. Главной целью перехода является получение возможности использовать новые свойства ОСРВ как в области защиты памяти (повышение надежности конечной системы), так и в области использования стандартных интерфейсов построения прикладного ПО (повторное использование программных компонентов, изначально разработанных для других проектов). Возможность подключения дополнительных программных компонентов без перезагрузки системы так же является привлекательной. Прежде всего необходимо определить программную архитектуру и набор свойств существующей системы управления, построенной на основе базовой ОСРВ. Предположим, что аппаратная платформа базируется на основе 32-разрядного процессора с тактовой частотой порядка нескольких сотен мегагерц и существует

несколько интерфейсов ввода-вывода с жесткими требованиями ко времени отклика системы. Также предположим, что прикладное ПО, обслуживающее интерфейсы ввода-вывода, разработано ранее на основе базовой ОСРВ.

С другой стороны, предположим, что такие программные компоненты, как расширенная поддержка стека протоколов TCP/IP, система поддержки оконного интерфейса с пользователем, система загрузки и запуска Java приложений разработаны для расширенных ОСРВ (например, QNX или Linux). Таким образом, стоит задача интеграции двух разнородных частей прикладного ПО – ПО, написанного для базовой ОСРВ, и ПО, написанного для расширенной ОСРВ.

Область применения систем варьируется от сетевых устройств (маршрутизаторы низшего класса) и систем автоматического сбора данных до систем управления, устанавливаемых в автомобили (телематические устройства (ТУ), бортовые компьютеры)). В случае ТУ интерфейсами ввода-вывода с жесткими требованиями ко времени отклика системы являются канал обмена данными между ТУ и внутренней сетью передачи данных автомобиля, каналы передачи звука и данных между ТУ и сотовым телефоном (как внутренним, так и внешним), канал обмена данными между центральным процессором и приемником сигналов от навигационных спутников.

Одноядерный и двуядерный подходы к построению систем на основе технологии Linux.

Задача интеграции разнородных частей прикладного ПО не является новой. Пути решения задач могут различаться в зависимости от требований на конечную систему, свойств и архитектуры заимствованного прикладного ПО, а также от свойств и архитектуры расширенной ОСРВ. Обсудим задачи по переносу ПО на систему Linux. В случае Linux задача интеграции разнородных частей прикладного ПО связана с задачей сокращения времени реакции системы Linux на входные воздействия и с задачей обеспечения гарантированного отклика системы Linux на входные воздействия в заданный промежуток времени. Как правило, различают два подхода (рис. 1) к решению задачи обеспечения гарантированного отклика системы Linux на входные воздействия в заданный промежуток времени – одноядерный и двуядерный.

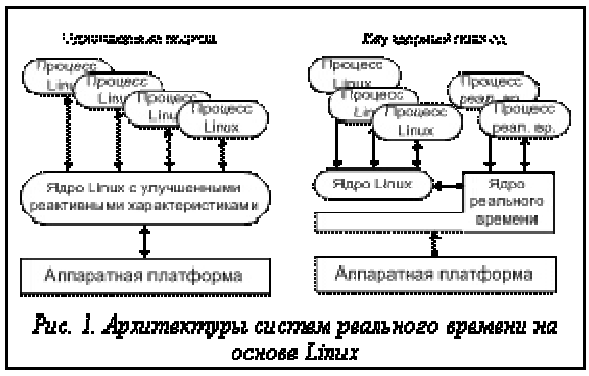


Рис. 1. Архитектуры систем реального времени на основе Linux

В первом случае (одноядерный подход) ядро Linux должно быть расширено средствами повышения скорости реакции системы на входные воздействия, так как стандартное ядро Linux не удовлетворяет ряду требований, предъявляемых к приложениям реального времени. Во втором случае (двуядерный подход) компактное ядро реального времени осуществляет диспетчеризацию задач реального времени и обработку прерываний от устройств ввода-вывода. Ядро Linux осуществляет диспетчеризацию задач Linux, не критичных ко времени реакции системы на входные воздействия. На первый взгляд кажется, что двуядерный подход позволит осуществить интеграцию наследованного прикладного ПО и нового прикладного ПО с наименьшими трудозатратами, так как не потребует переработки прикладного ПО, связанной с заменой нестандартных вызовов сервисов базовой ОСРВ на вызовы, поддерживаемые ядром Linux. Кроме того, двуядерный подход дает гарантию, что реактивные характеристики заимствованного прикладного ПО не ухудшатся после переноса системы на новую программную платформу, так как ядро ОСРВ по-прежнему будет иметь наибольший приоритет по сравнению с ядром Linux.

Интеграция заимствованного прикладного кода ОСРВ с ядром Linux в рамках двуядерного подхода.

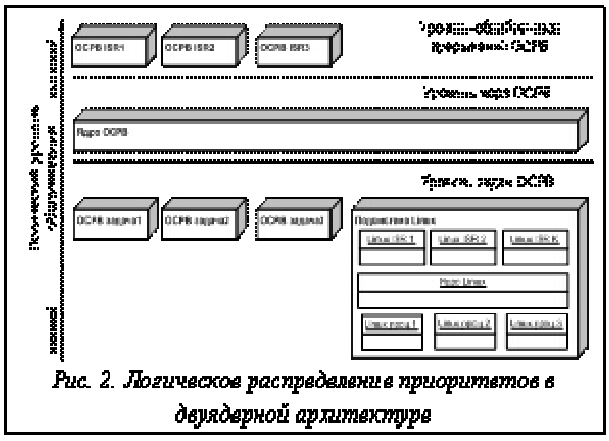


Рис. 2. Логическое распределение приоритетов в двуядерной архитектуре

На рисунке 2 представлена схема распределения логических приоритетов в двуядерной архитектуре ПО, в которой обработчики прерываний ОСРВ, ядро ОСРВ и задачи ОСРВ имеют приоритет исполнения больший, нежели обработчики прерываний Linux, ядро Linux и процессы, работающие под управлением Linux.

Подобное логическое распределение приоритетов позволяет гарантировать неизменность времени реакции заимствованного прикладного ПО ОСРВ на входные воздействия. Необходимо отметить, что предложенная схема распределения приоритетов является идеальной и не всегда может быть реализована на нужных аппаратных платформах. Как правило, трудности возникают в области обработчиков прерываний, относящихся к подсистеме Linux. С одной стороны, необходимо обеспечить целостность данных, с которыми приходится иметь дело в обработчиках прерываний Linux. С другой стороны, необходимо приостанавливать исполнение кода обработчиков прерываний Linux при возникновении запросов на обработку прерываний, логически относящихся к подсистеме ОСРВ. Более того, необходимость приостановки исполнения кода обработчиков прерываний Linux появляется и при возникновении запросов на запуск задач ОСРВ. В некоторых случаях приходится иметь дело с так называемым эффектом инверсии приоритетов в области обработчиков прерываний Linux и задач ОСРВ. Иногда удается найти частное решение данной проблемы, используя архитектурные особенности целевой аппаратной платформы и заимствованной ОСРВ. В иных случаях применима техника эмуляции аппаратных прерываний, обработчики которых находятся в подсистеме Linux.

Интеграция двух операционных систем (заимствованной ОСРВ и системы Linux) требует внесения некоторых изменений в исходный код обеих систем. К счастью, некоторые изменения, связанные с интеграцией высокоприоритетного ядра ОСРВ, уже сделаны в ядрах Linux версий 2.4.XX и выше. Существует механизм встраивания вызовов функций внешнего кода из кода инициализации ядра Linux, из кода обработки прерываний и непосредственно из ядра. Более того, существует возможность замены процедур аппаратного запрещения и разрешения прерываний в ядре Linux на соответствующие программные механизмы. Перечисленные свойства ядра Linux могут быть использованы при его интеграции с заимствованной ОСРВ.

Со стороны ядра заимствованной ОСРВ необходимо, как правило, изменить части кода, отвечающие за распределение физической и виртуальной памяти в системе.

Необходимо отметить, что схема распределения памяти (рис. 3) – частный пример. В некоторых реализациях двуядерной архитектуры ядро ОСРВ работает непосредственно в области логических адресов Linux и представляет собой загружаемый модуль ядра Linux.

Кроме задачи распределения памяти между подсистемами ОСРВ и Linux обычно стоит задача организации взаимодействия между частями двуядерной системы. Следует выделить три группы функций по организации взаимодействия между частями двуядерной системы:

- по передаче загрузочной информации между ядром ОСРВ и ядром Linux (например, распределение физической и виртуальной памяти);
- по обмену данными между задачами ОСРВ и приложениями Linux;
- по синхронизации исполнения задач ОСРВ и приложений Linux.

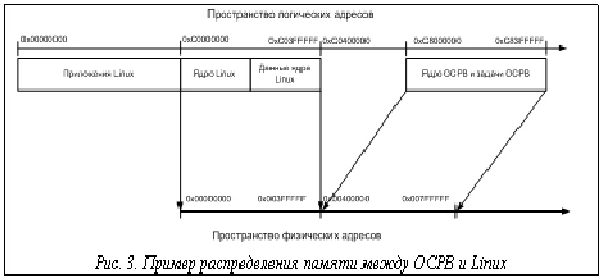


Рис. 3. Пример распределения памяти между ОСРВ и Linux

Как правило, необходимо добавить модули по организации межъядерного взаимодействия как в ядро ОСРВ, так и в ядро Linux. Отметим, что наибольшую трудность вызывает разработка механизмов синхронизации исполнения задач реального времени и приложений Linux.

Некоторые достоинства и недостатки двуядерного подхода при построении приложений реального времени на основе Linux те. Необходимо отметить, что использование двуядерного подхода позволяет достичь быстрых практических результатов, когда стоит задача разработки демонстрационных версий продукта с ограниченным набором функций, критичных ко времени исполнения. Как правило, достаточно легко запустить усеченную версию наследованного прикладного ПО ОСРВ и интегрировать ПО с подсистемой Linux. В подобной конфигурации возможна организация передачи данных из прикладного ПО ОСРВ в подсистему Linux для дальнейшей обработки и визуализации [5]. Это является несомненным преимуществом двуядерного подхода как подхода, пригодного для разработки демонстрационных версий продукта. Однако, как правило, запуск полной версии наследованного прикладного ПО ОСРВ, критичного ко времени исполнения, вызывает существенные затруднения. Эти затруднения связаны как с проблемой разделения ресурсов центрального процессора и периферийных устройств между ядрами ОСРВ и Linux, так и с существенным затруднением процесса отладки системы. Для отладки двуядерной системы необходимы большие усилия, так как локализация проблем требует анализа взаимодействия между разнородными частями системы, что, как правило, не является тривиальной задачей. Более того, стандартные отладчики не поддерживают отладку двуядерных систем в терминах объектов операционных систем. Процесс анализа производительности систем затруднен в силу тех же причин. Таким образом, кажется разумным рекомендовать двуядерный Linux подход к построению систем реального времени с интегрированным заимствованным кодом прикладного ПО, если стоит задача разработки прототипов ПО и демоверсий ПО с ограниченным набором функций, критичных ко времени исполнения. Одноядерный подход и коммерческие операционные системы (например QNX) наиболее привлекательны при разработке полнофункциональных версий продуктов с существенными объемами заимствованного кода, критичного ко времени исполнения. Очевидно, что задача сохранения заданных реактивных характеристик системы является первоочередной при использовании одноядерного подхода.

В заключение отметим, что мы кратко рассмотрели круг задач по переносу прикладного ПО с базовых ОСРВ, таких как OSEK и RTXC, на расширенные ОСРВ, такие как Linux. Также рассмотрены одноядерный и двуядерный подходы к разработке систем управления реального времени на основе Linux технологии. Двуядерный Linux подход рассмотрен более подробно с точки зрения интеграции ОСРВ с ядром Linux.

Ключевые вопросы:

1. Существующие типы интеграции ПО.

Литература:

1 Избачков Ю.С. Информационные системы: учеб.:рек. Мин. обр. и науки РФ/ Ю. С. Избачков, В. Н. Петров. -2-е изд. -СПб.: Питер, 2008.-656 с.

2 Гвоздева Т.В. Проектирование информационных систем: учеб. пособие: рек. УМО/ Т.В. Гвоздева, Б.А. Баллод. -Ростов н/Д: Феникс, 2009.-509 с.

3 Хомоненко А. Д. Базы данных: учеб.: рек. УМО / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев; под ред. А. Д. Хомоненко. -6-е изд., доп. -СПб.: КОРОНА-Век, 2009.-736 с.

Практическое занятие 9. Место интеграции ПО в жизненном цикле предприятия.

Неформальный подход, применяющийся к построению некоторых программ, недостаточен для разработки больших систем. Стоимость аппаратных средств постепенно снижается, тогда, как *стоимость программных продуктов стремительно возрастает*. Возникла необходимость в новых технологиях и методах управления комплексными проектами разработки больших программных систем. Такие методы составили *программную инженерию*. Возрастает как объем производства программного продукта, так и его сложность. Кроме того, сближение вычислительной и коммуникационной техники ставит новые требования перед специалистами. Это также является одной из причин возникновения проблем при разработке программных систем, как и то, что многие компании, занимающиеся производством

ПС, не уделяют должного внимания эффективному применению современных методов и стандартов, разработанных в программной инженерии.

Программная инженерия — как часть системотехники охватывает все аспекты *жизненного цикла ПС* от начальной стадии разработки системных требований до завершения использования программного продукта. При этом специалисты выполняют практическую, инженерную работу. Они применяют теоретические построения, методы и средства там, где это необходимо, но делают это выборочно и всегда пытаются найти практическое решение задачи, даже если не существует подходящей теории или методов решения. Инженеры всегда должны понимать, что они работают в организационных и финансовых рамках заключенных контрактов, и ищут решение поставленной перед ними задачи *с учетом условий контракта*. Программная инженерия не рассматривает технические аспекты детального создания компонентов — в её ведение входят такие задачи, как управление проектами ПС и разработка средств, методов и теорий, необходимых для обеспечения жизненного цикла комплексов программ. Программирование компонентов — это дело, главным образом, индивидуальное, а программная инженерия систем — всегда *коллективная работа*.

Программные средства все больше встраиваются в различные системы. Работа с такими проектами требует от программного инженера широкого взгляда на общие *задачи проектирования систем*. Программному инженеру необходимо участвовать в выработке требований для всей системы, а также пытаться понять прикладную область ПС еще до начала обдумывания абстрактных интерфейсов, требованиям которых должен будет отвечать программный продукт. Рассматривая программную инженерию *как часть системотехники*, обнаруживается *важность компромисса* как отличительного признака любой инженерной дисциплины. Существуют принципиальные трудности изменения масштаба при попытке привнести приемы написания малых программ в проектирование больших программных комплексов.

Разработчики проекта системы вынуждены тратить время на общение друг с другом, вместо того, чтобы писать программы. Иногда люди покидают проект, и это влияет не только на работу, выполняемую непосредственно ими, но и на работу тех, кто от них зависит. Замена разработчика в проекте может требовать обучения и серьезнейшей подготовки нового специалиста для освоения им технических условий проекта и текущего состояния системы. Любое изменение первоначальных требований к системе влияет на многие составные части проекта, выливаясь в дальнейшем в задержку поставки готового продукта. Как в любой инженерной отрасли, программный инженер должен развивать умения, позволяющие построить набор моделей и оценить эти модели, управляя выбором компромиссов. Такие модели используются на этапе определения требований к проектируемой системе, в разработке архитектуры программного средства и на стадии реализации проекта. Программный инженер — это *член команды*, поэтому должен обладать навыками общения и межличностных отношений, а также уметь планировать не только свою работу, но и координировать её с работой других.

Специалист по программной инженерии должен знать системотехнику вычислительных систем, поскольку здесь программный компонент играет определяющую роль. Таким образом, технологии программной инженерии часто являются критическим фактором при разработке сложных вычислительных систем. Интеграционные свойства систем проявляются только тогда, когда система рассматривается как единое целое. В этом состоит сложность прогнозирования и оценки её свойств, поскольку иногда можно измерить характеристики только подсистем, из которых состоит комплексная система. Высокие темпы роста основных ресурсов аппаратных средств (приблизительно на порядок каждые пять лет), и сохраняющаяся потребность в увеличении их использования со стороны различных пользователей и сфер применения, приводят к необходимости *адекватного совершенствования технологий создания программных средств и баз данных*.

Ключевые вопросы:

1. Причины возникновения проблем при разработке программных систем;

2. Программная инженерия — как часть системотехники;
3. Интеграционные свойства систем.

Литература:

1 Избачков Ю.С. Информационные системы: учеб.:рек. Мин. обр. и науки РФ/ Ю. С. Избачков, В. Н. Петров. -2-е изд. -СПб.: Питер, 2008.-656 с.
2 Гвоздева Т.В. Проектирование информационных систем: учеб. пособие: рек. УМО/ Т.В. Гвоздева, Б.А. Баллод. -Ростов н/Д: Феникс, 2009.-509 с.
3 Хомоненко А. Д. Базы данных: учеб.: рек. УМО / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев; под ред. А. Д. Хомоненко. -6-е изд., доп. -СПб.: КОРОНА-Век, 2009.-736 с.

3. Методические материалы к выполнению лабораторных работ

Лабораторная работа 1. Интегрированные распределенные приложения.

Цель работы: Познакомиться с общими принципами организации распределенных ИС. Получить представление о компонентной объектной модели (COM). Научиться использовать механизм автоматизации для обработки данных с использованием внешних приложений.

Указания к выполнению лабораторной работы

Классическая компьютерная информационная система включает в себя следующие обязательные элементы:

Информационная база, которая является основным хранилищем данных ИС. В подавляющем большинстве случаев она представляет собой*серверную базу данных*, которая управляется некоторой СУБД. Иные варианты (например, совместно используемый набор файлов) встречаются значительно реже.

Клиентские приложения, которые используются для работы с данными и устанавливаются на ЭВМ пользователей информационной системы. Клиентские приложения не только выполняют просмотр и редактирование данных (так называемые *презентационные функции* информационной системы), но и обеспечивают решение более сложных задач: контроль корректности вводимой информации, статистическая обработка данных, формирование отчетов и т.п. (*функции ИС по прикладной обработке данных*).

Средства обеспечения доступности данных информационной базы из клиентских приложений. К таким средствам, прежде всего, относятся средства сетевого доступа, которые обычно реализованы соответствующими средствами сетевых операционных систем (например, поддержка протоколов TCP/IP). Кроме того, сюда следует отнести средства взаимодействия пользовательского приложения и сервера баз данных. Некоторые из них были изучены в ходе выполнения предыдущих лабораторных работ (процессор баз данных BDE, технологии ODBC и ADO и т.п.).

При эксплуатации описанных информационных систем нередко возникают следующие проблемы:

Необходимость использования ЭВМ с высокими требованиями к аппаратной части при организации рабочих мест пользователей. Клиентские приложения, которые реализуют не только презентационные, но и прикладные функции ИС, обычно требуют для своей работы довольно значительных вычислительных ресурсов. Кроме того, дополнительные ресурсы необходимы для установки и использования средств обеспечения доступа к данным.

Необходимость приобретения (лицензирования) и установки специальных средств доступа к данным (например, компонентов BDE или поставщиков данных OLE DB) на каждом компьютере пользователя ИС.

Сложность конфигурирования средств доступа к данным (например, организация BDE-псевдонимов или настройка источников данных ODBC на всех пользовательских ЭВМ).

Возможные конфликты между средствами доступа к данным, установленным для работы с различными информационными системами.

Указанные проблемы могут существенно повысить стоимость эксплуатации ИС и значительно уменьшить (или вовсе устранить) экономический эффект от ее внедрения. Для пре-

одоления этих проблем в последние годы все чаще рекомендуется проектировать сложные информационные системы на базе многоуровневой (многозвенной) архитектуры.

Рассмотренная классическая структура информационной системы соответствует двухуровневой модели клиент-сервер. Клиентами в данном случае являются приложения, установленные на ЭВМ пользователей, а сервером – очевидно, сервер базы данных (СУБД либо сетевая ОС – если база данных состоит из множества совместно используемых файлов). Клиенты напрямую обращаются к серверу (посредством служебных средств обеспечения доступа к данным) и единственной информационной услугой, которую они запрашивают, являются хранящиеся в информационной базе данные. Большая часть прикладной обработки данных выполняется клиентским приложением самостоятельно.

Использование трехуровневой архитектуры ИС позволяет разделить функции представления и прикладной обработки данных, а также упростить и унифицировать доступ к данным из клиентских приложений. Это достигается путем выделения части функций информационной системы в отдельный промежуточный слой. В этом случае структура ИС будет состоять не из трех, а из четырех элементов:

Данные по-прежнему хранятся в информационной базе, в роли которой чаще всего выступает серверная БД.

Функции прикладной обработки данных сосредоточены в так называемом *сервере приложений* (*Application Server*), который предоставляет клиентским пользовательским приложениям множество информационных услуг (сервисов) высокого уровня. Серверы приложений выступают в качестве промежуточного уровня между серверами данных и клиентскими приложениями. С одной стороны, они взаимодействуют с информационной базой ИС, запрашивая необходимые данные и отсылая запросы на модификацию данных. В этом случае они выступают в качестве клиентов. С другой стороны, они взаимодействуют с клиентскими пользовательскими приложениями, выдавая по их запросам результаты обработки данных или транслируя полученные запросы на модификацию данных в информационную базу. В этом случае они выступают в качестве серверов. Серверы приложений являются объектами высокого уровня, поэтому их реализация может не зависеть от специфики конкретной операционной системы или СУБД. Это обеспечивает целый ряд важных достоинств многоуровневой архитектуры ИС, в том числе:

Возможность подключения к серверу приложений клиентских приложений, работающих под управлением различных ОС.

Возможность простого и единообразного подключения клиентских приложений. Для их взаимодействия с сервером приложений может быть использована простая, но достаточно эффективная технология, не требующая установки сложного программного обеспечения на пользовательские ЭВМ. Все средства доступа к данным информационной базы (менеджеры драйверов, поставщик данных для конкретной СУБД, библиотеки, реализующие прикладной программный интерфейс и т. п.) в этом случае сосредоточены на уровне сервера приложений, а не на уровне клиентов.

Клиентские приложения реализуют только *презентационные функции* ИС (просмотр и редактирования данных). Это соответствует концепции так называемого «тонкого» клиента, предъявляющего минимальные требования к аппаратной части пользовательских ЭВМ.

Средства обеспечения доступности данных теперь распределены между клиентами и серверами приложений. Средства сетевого взаимодействия по-прежнему необходимы на всех уровнях, однако средства взаимодействия с базами данных необходимы только на уровне серверов приложений (см. выше).

Развитием трехуровневой архитектуры является так называемая *многоуровневая* (*n-уровневая*) *организация вычислений* (*Multi-tier computing*), когда информационная система состоит из большого количества удаленных друг от друга объектов (серверов), каждый из которых может предоставлять другим объектам (клиентам) разнообразные информационные услуги. При этом различные серверы могут работать под управлением различных ОС или использовать для хранения данных СУБД различных типов. Для синхронизированного и со-

гласованного функционирования таких объектов в состав информационной системы также включают специальные служебные сервисы – системы мониторинга и управления распределенными транзакциями, агенты запуска и управления серверами и др.

В настоящее время наиболее популярны следующие технологии организации распределенных информационных систем:

Технология COM и ее развитие (DCOM, COM+);

Технология MIDAS (на базе компонентной объектной модели COM);

Технология CORBA;

Платформа .NET.

Сокращение COM означает Component Object Model, что можно перевести с английского языка как компонентная объектная модель. Технология была разработана фирмой Microsoft в начале 90-ых годов прошлого века для обеспечения взаимодействия между различными приложениями, запущенными на одном или даже на разных компьютерах.

Сущность технологии COM заключается в программировании с использованием *компонентов* – подход, который знаком всем программистам, использующим в своей работе среду разработки Delphi или C++ Builder. Под компонентом в данном случае понимается законченный (и откомпилированный) объект со своими свойствами и методами, который может легко встраиваться в различные приложения и распространяться как отдельный продукт. Компоненты, созданные в соответствии со спецификацией COM, могут функционировать в различной языковой и операционной средах. Это значит, что если разработчик оформил некоторый набор функций как объект COM, то функциями этого объекта могут воспользоваться программисты самых разных языков программирования: C++, Delphi, Visual Basic и т. д. – достаточно, чтобы соответствующая среда разработки поддерживала технологию COM. По этой причине модель COM может являться базовой для создания распределенных информационных систем – составляющие ее объекты могут быть реализованы с использованием различных технологий и инструментов программирования, однако их взаимодействие может осуществляться в соответствии со спецификацией COM (посредством определенных интерфейсов и протоколов).

Принципы обращения к свойствам и методам COM-объекта из других приложений полностью соответствуют модели клиент-сервер. Компонент, в котором реализованы некоторые полезные свойства и методы, выступает в качестве COM-сервера, а обращающиеся к нему приложения – в качестве клиентов. Одно и то же приложение, очевидно, может выступать и в качестве клиента, и в качестве COM-сервера, в зависимости от характера взаимодействия с другими приложениями в каждый конкретный момент времени. Чтобы взаимодействие «клиент-сервер» между произвольной парой приложений состоялось успешно, необходимо выполнение следующих условий:

Приложение-клиент должно «иметь представление» о тех сервисах (полезных свойствах и методах), которые предоставляет приложение-сервер. Если бы речь шла о приложениях, написанных на одном языке программирования, то для решения этой проблемы достаточно было бы распространять вместе с компонентом описания соответствующих ему классов. Например, для компонента, написанного на C++, описания его классов хранятся в заголовочных файлах с расширением “*.h*”. Поскольку технология COM является не зависимой от языка программирования, то для определения сервисных функций COM-объекта используется специальный язык – *язык описания интерфейса*(*IDL*, *Interface Description Language*).

Все COM-объекты должны иметь уникальные имена, по которым их можно отличать друг от друга и обращаться к ним. При этом уникальность должна носить глобальный характер, поскольку одни и те же COM-объекты могут использоваться десятками тысяч разработчиков в сотнях тысяч различных приложений. Для именования объектов COM используются так называемые глобальные уникальные идентификаторы (*Globally unique identifier*, *GUID*[3]). Глобальный уникальный идентификатор представляет собой 128-разрядное число, которое генерируется с использованием алгоритмов получения случайных чисел и специальной хеш-функции. Вероятность повторения глобального уникального идентификатора такой разряд-

ности чрезвычайно мала. Таким образом, для того чтобы все COM-серверы были поименованы уникальными именами, разработчикам достаточно использовать один и тот же заранее определенный алгоритм генерации глобальных уникальных идентификаторов. В ОС Windows для генерирования GUID используются функции API **UuidCreate** и **CoCreateGuid**. Глобальный уникальный идентификатор принято записывать в следующем формате:

XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX,

например:

B502D1BE-9A57-11d0-8FDE-00C04FD9189D

Операционная система должна хранить список доступных COM-объектов и информацию о месте их хранения (то есть путь к соответствующим исполняемым модулям, в которых реализованы сервисы этих объектов). В ОС Windows для этих целей используется специальный подраздел системного реестра: HKEY_CLASSES_ROOT\CLSID\, у которого в качестве вложенных подразделов выступают имена (глобальные уникальные идентификаторы) зарегистрированных в системе объектов, например:

{00000000-0E4D-0463-87B5-D411BEBE0010}
{00000001-4FEF-40D3-B3FA-E0531B897F98}
{00000010-0000-0010-8000-00AA006D2EA4}
{00000100-0000-0010-8000-00AA006D2EA4}

Каждый из указанных разделов содержит описание свойств соответствующего COM-объекта. Например, свойство **ProgID** содержит программный идентификатор объекта – кодовое наименование, которое присваивается разработчиком и которое, в отличие от глобального уникального идентификатора, несет некоторую смысловую нагрузку. Чтобы задать местонахождение исполняемого модуля для объекта COM используются свойства **InprocServer32** (если реализация объекта располагается в DLL-файле), **LocalServer32** (если объект реализован как EXE-файл) или **RemoteServer32** (для объектов, расположенных на удаленных ЭВМ в компьютерной сети).

Операционная система, под управлением которой работает COM-сервер, должна иметь возможность единообразного получения адреса зарегистрированного COM-объекта, чтобы передавать соответствующую ссылку запросившим ее клиентам. Кроме того, для взаимодействия клиента с удаленными COM-серверами необходим специальный протокол сетевого взаимодействия, реализующий, в том числе, правила аутентификации и авторизации.

На основании указанного выше можно сделать вывод, что создание полноценных COM-объектов – это весьма сложная задача, требующая от разработчика глубоких знаний в области системного и объектно-ориентированного программирования.

Среда разработки Borland C++ Builder предоставляют программисту возможность не только создавать собственные, но и эффективно использовать уже существующие COM-объекты в своих приложениях. В некоторых случаях взаимодействие создаваемого приложения с объектами COM реализуется незаметно («прозрачно») для разработчика, в других – требует от него осознанных действий.

Полезным примером практического и осознанного использования технологии COM является использование *механизма автоматизации* для взаимодействия с внешними приложениями. Механизм автоматизации (*Automation*[5]) позволяет разработчикам приложения привлекать для обработки данных функциональные возможности других приложений. Например, для формирования отчетов часто используются возможности Microsoft Excel или Microsoft Word.

Взаимодействие в данном случае осуществляется между *Объектами* и *контроллерами* автоматизации. Объект автоматизации – это объект, созданный по технологии COM, в котором реализован доступ к полезным свойствам и методам некоторого приложения или его части. Контроллер автоматизации – это приложение, которое имеет доступ к свойствам и методам COM-объекта и использует их для обработки своих данных. В указанном ранее примере, приложение разработчика – это контроллер, а Microsoft Excel либо Microsoft Word – объекты автоматизации.

Рассмотрим общую схему работы контроллера автоматизации при взаимодействии с каким-либо приложением Microsoft Office:

1. Проверить, запущена ли копия приложения-сервера (объекта автоматизации).
2. В зависимости от результатов проверки (либо исходя из конкретной ситуации) запустить копию приложения-сервера либо подключиться к уже имеющейся копии.
3. Если необходимо, сделать окно приложения-сервера видимым.
4. Выполнить какие-то действия с приложением-сервером (например, создать или открыть документы, изменить их данные, сохранить документы и т. п.).
5. Закрыть приложение-сервер, если его копия была запущена данным контроллером, или отключиться от него, если контроллер подключился к уже имеющейся копии (или если работа с приложением будет продолжена).

Ниже приведен программный код для указанной схемы (в качестве объекта автоматизации выбрано приложение Microsoft Excel, которое запускается по нажатию на кнопку Button1 и закрывается по нажатию кнопки Button2):

```
#include
...
TForm1 *Form1;
// глобальные переменные модуля
Variant Server;
bool ServerIsRunning;
...
void __fastcall TForm1::Button1Click(TObject *Sender)
{
AnsiString ServerProgID;
IUnknown *UnknownInterface;
HResult Res;
// Свойство ProgID для приложения Microsoft Excel = "Excel.Application"
ServerProgID = "Excel.Application";
// Проверяем наличие запущенной копии приложения
Res = GetActiveObject(ProgIDToClassID(ServerProgID),0,&UnknownInterface);
if SUCCEEDED(Res)
{
// подключаемся к существующему экземпляру приложения-сервера
Server = GetActiveOleObject(ServerProgID);
ServerIsRunning = true;
}
else
{
// запускаем новый экземпляр приложения-сервера
Server = CreateOleObject(ServerProgID);
ServerIsRunning = false;
}
// Показываем окно приложения-сервера на экране
Server.OlePropertySet("Visible", true);
try
{
//----------------------------------------
// Здесь выполняются некоторые действия
// с объектами приложения Office
// ----------------------------------------
}
catch(...)
```

```
{
// Сообщение об ошибке работы с MS Excel
}
Button1->Enabled = false;
Button2->Enabled = true;
}
//---------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
// отключаемся либо закрываем приложение-сервер
if (ServerIsRunning)
Server= Unassigned;
else
Server.OleProcedure("Quit");
Button2->Enabled = false;
Button1->Enabled = true;
}
```

Для обращения к свойствам и методам объектов автоматизации можно использовать следующие методы класса Variant:

- Variant OlePropertyGet(const String& name, TAutoArgsBase* args = 0) – получить значение свойства *name* некоторого COM-объекта. В качестве второго параметра метода обычно передаются дополнительные аргументы;
- **void** OlePropertySet(const String& name, TAutoArgsBase& args) – установить значение свойства *name* некоторого COM-объекта. В качестве второго параметра метода обычно передается устанавливаемое значение;
- **void** OleProcedure (const String& name, TAutoArgsBase* args = 0) – запустить метод (процедуру) с именем *name* для некоторого COM-объекта. В качестве второго параметра метода обычно передаются дополнительные параметры процедуры;
- Variant OleFunction (const String& name, TAutoArgsBase* args = 0) – запустить метод (функцию) с именем *name* для некоторого COM-объекта. В качестве второго параметра метода обычно передаются дополнительные параметры функции;

Используя перечисленные методы, можно обращаться к тем объектам приложения Microsoft Excel (Workbooks, WorkSheets, Cells и т. п.), которые изучались в лабораторной работе № 2 курса «Информационные системы».

Примеры:

1. Открываем новую рабочую книгу

```
Server.OlePropertyGet("WorkBooks").OleProcedure("add");
```

2. Устанавливаем ширину колонки “B” равной 40 на первом рабочем листе:

```
Variant Sheet1 = Server.OlePropertyGet("WorkSheets",1);
Variant ColumnB =
Sheet1.OlePropertyGet("Cells",1,2).OlePropertyGet("EntireColumn");
ColumnB.OlePropertySet("ColumnWidth",40);
```

3. Устанавливаем значение ячейки с RC-адресом (Row, Column) равным значению переменной (или Variant-объекта) Data:

```
Sheet1.OlePropertyGet("Cells",Row,Column).OlePropertySet("Value",Data);
```

4. Устанавливаем жирный шрифт для ячейки с RC-адресом (Row, Column)

```
Sheet1.OlePropertyGet("Cells",Row,Column).
OlePropertyGet("Font").OlePropertySet("Bold",true);
```

5. Закрываем последнюю открытую рабочую книгу. Если в нее были внесены изменения, то пользователь будет предупрежден об их возможной потере.

```
Variant WorkBook = Server.OlePropertyGet("WorkBooks",
Server.OlePropertyGet("WorkBooks").OlePropertyGet("Count"));
WorkBook.OleProcedure("close");
```

6. Закрываем последнюю открытую рабочую книгу без сохранения изменений и без предупреждения пользователя.

```
Variant WorkBook = Server.OlePropertyGet("WorkBooks",
Server.OlePropertyGet("WorkBooks").OlePropertyGet("Count"));
WorkBook.OleProcedure("close",false);
```

^ Задания к лабораторной работе

1. Исследовать перечень COM-объектов, зарегистрированных на компьютере, на котором выполняется лабораторная работа. Для этого запустить редактор системного реестра (это можно, сделать, например, через команду **Выполнить** кнопки **Пуск**; имя приложения редактора реестра – *regedit.exe*). В редакторе реестра найти раздел HKEY_CLASSES_ROOT и подраздел CLSID. В отчет о выполнении лабораторной работы включить рисунок с частью системного реестра, отображающий перечень нескольких первых объектов COM.
2. Найти в списке зарегистрированных COM-объектов объекты со следующими глобальными уникальными идентификаторами:

{00000011-0000-0010-8000-00AA006D2EA4}
{00000100-0000-0010-8000-00AA006D2EA4}
{00024500-0000-0000-C000-000000000046}

В отчете о выполнении лабораторной работы указать, какие из указанных объектов были обнаружены. Для найденных объектов привести значения свойств **ProgID** и **InprocServer32**/**LocalServer32**/**RemoteServer32**.

3. С использованием механизма автоматизации доработать БД-приложение из лабораторной работы № 5 таким образом, чтобы у пользователя была возможность экспортировать в Microsoft Excel данные из основной таблицы приложения. Обратить внимание на удобство восприятия экспортированных данных (предусмотреть выделение заголовков столбцов, подбор ширины ячеек и т.п.).
4. В отчете отразить ход выполнения работы, результаты тестирования и сделанные выводы.

Контрольные вопросы и задания

1. Двухзвенная архитектура ИС.
2. Достоинства и недостатки двухзвенной архитектуры.
3. Особенности трехзвенной архитектуры ИС.
4. Распределенные (многозвенные) ИС.
5. Принципы технологии COM.
6. Понятие глобального уникального идентификатора.
7. Механизм автоматизации как средство межпроцессного взаимодействия.
8. Общая схема работы контроллера автоматизации.
9. Приведите примеры «прозрачного» использования механизма автоматизации при разработке приложений в Borland C++ Builder.
10. Функции и методы для взаимодействия с объектами автоматизации в Borland C++ Builder.

Лабораторная работа 2. Протокол ODBC и его реализации.

Цель работы: Изучить принципы разработки приложений для работы с БД в Borland C++ Builder. Получить навыки работы с технологией ODBC.

Указания к выполнению лабораторной работы

Процессор баз данных Borland Database Engine – не единственный механизм организации доступа к данным в БД-приложениях. Разумеется, другие производители программного обеспечения также занимались разработкой универсальных средств взаимодействия с СУБД

различных производителей. Важным примером таких разработок является технология ODBC, которая на сегодняшний день стала фактическим отраслевым стандартом работы с базами данных из клиентских приложений.

Аббревиатура ODBC расшифровывается как *Open DataBase Connectivity*, что можно перевести как «открытая система связи с базами данных». Признак «открытости» в данном случае означает простоту расширяемости, наращиваемости системы, то есть возможность легко использовать эту технологию для работы с новой, еще одной СУБД.

Архитектура ODBC во многом аналогична ранее рассмотренной архитектуре BDE. В системе взаимодействия приложений с базами данных посредством ODBC принято выделять следующие компоненты:

- Приложения, разработанные с использованием *прикладного программного интерфейса ODBC* (ODBC API). Прикладной программный интерфейс ODBC представляет собой описание процедур и функций, обеспечивающих стандартные операции по работе с базами данных: подключение и регистрацию в СУБД, выполнение операторов на языке манипулирования данными SQL, извлечение информации из БД и ее представление в виде *наборов данных* в памяти приложения. Кроме этого, в интерфейс ODBC входят стандартное представление для данных различных типов, стандартный набор кодов ошибок и типовой синтаксис SQL-операторов, которые можно использовать для обращения к СУБД.
- *Менеджер драйверов* (ядро ODBC). Он представляет собой библиотеку, в которой содержится реализация упомянутого выше интерфейса ODBC. Менеджер драйверов является независимым по отношению к приложениям и к СУБД и выступает в качестве транслятора между приложением и сервером конкретной базой данных. Его основная функция состоит в загрузке (по требованию приложения) подходящего драйвера для той СУБД, с которой приложение собирается взаимодействовать.
- ODBC *драйверы* и *агенты БД*. Эти компоненты обрабатывают вызовы функций ODBC и направляют запросы на языке SQL к *источникам данных* (базам данных конкретных СУБД), а также возвращают полученные результаты приложению. При необходимости драйверы выполняют модификацию исходного запроса приложения таким образом, чтобы он соответствовал синтаксическим требованиям целевой СУБД.
- *Источники данных*. Содержат те данные, доступ к которым необходим пользователю приложения. Данные сохраняются в базах данных, контролируемых некоторой СУБД, операционной системой, а также сетевой операционной системой, если таковая используется.

В качестве аналогов BDE-псевдонимам в рассматриваемой технологии выступают *поименованные источники данных* (Data Source Names, DSN) – специальные структуры данных, которые используются для описания соединений с конкретными базами данных. Важнейшими параметрами любого поименованного источника данных является ODBC-драйвер, который будет использоваться для обращения к СУБД, и местонахождение целевой БД (путь к файлу, адрес и наименование сервера баз данных и т.п.).

Управление источниками данных и настройка всей системы ODBC осуществляется с использованием ODBC-администратора. В семействе ОС Windows таковым является специальная программа odbcad32.exe, которую можно вызвать из панели управления. Например (для Windows 2000 и Windows XP): **Пуск | Настройка | Панель управления | Администрирование | Источники данных (ODBC)**.

Рассмотрим пример создания поименованного источника данных для базы данных Microsoft Access.

1. В окне ODBC-администратора нас в первую очередь будет интересовать вкладка «Драйверы», которая предоставляет список установленных на компьютере драйверов ODBC. Стандартная установка системы ODBC фирмы Microsoft включает в себя драйверы ODBC для таких популярных СУБД как Micro-

soft SQL Server, Microsoft Access, а также драйвер ODBC для обращения к таблицам Microsoft Excel. Необходимо убедиться, что драйвер для работы с нужной нам СУБД присутствует в списке (например, строчкой «Microsoft Access Driver (*.mdb)»).

2. Списки поименованных источников данных находятся на вкладках «Пользовательский DSN» и «Системный DSN». Разница между ними в том, что пользовательские источники данных используются текущем пользователем компьютера, в то время как системные – доступны любому пользователю этого компьютера при условии наличия у него необходимых прав. На вкладке «Пользовательский DSN» нажмем кнопку «Добавить».
3. В окне «Создание нового источника данных» необходимо выбрать драйвер для работы с источником данных – в нашем случае «Microsoft Access Driver (*.mdb)». Для продолжения операции следует нажать кнопку «Готово».
4. В окне настройки поименованного источника данных следует задать следующие обязательные параметры:
    - Имя источника данных (например, «MYACCESS_ODBC»);
    - База данных – для этого необходимо нажать кнопку «Выбрать…» и указать путь и имя файла, в котором хранится нужная база данных Microsoft Access.

Если все обязательные параметры заданы, то кнопка «OK» в текущем окне станет доступной. Ее нажатие приведет к сохранению нового поименованного источника данных.

Следует помнить, что принцип создания и модификации поименованных источников данных ODBC не зависит от типа целевой СУБД. Однако внешний вид окна настройки, а также список обязательных для заполнения параметров может изменяться. Например, для настройки источника данных для СУБД Sybase SQL Server необходимо обязательно задавать имя источника данных, имя сервера баз данных и способ взаимодействия драйвера ODBC и СУБД (разделяемая память, поименованные каналы, стеки протоколов TCP/IP или IPX/SPX).

К достоинствам технологии ODBC на сегодняшний день можно отнести:

- Независимость стандартного интерфейса ODBC от языка программирования, типа СУБД и типа операционной системы. На сегодняшний день реализованы системы ODBC для ОС Microsoft Windows, а также для различных версий ОС Unix и ОС Linux.
- Открытость стандарта ODBC и наличие средств разработки новых драйверов. К настоящему моменту созданы сотни ODBC-драйверов для различных СУБД.
- Развитие и поддержка стандарта крупнейшими производителями программного обеспечения (в первую очередь, фирмой Microsoft).

Недостатками технологии ODBC являются:

- Многие возможности современных СУБД не поддерживаются стандартом ODBC;
- Многослойная архитектура ODBC может вызвать проблемы снижения производительности БД-приложений.
- Открытость стандарта привела к появлению огромного количества драйверов низкого качества, использование которых приводит к возникновению многочисленных ошибок и сбоев в работе.
- Технология ODBC не позволяет прозрачно работать с распределенными по различным СУБД источниками данных.

Контрольные вопросы

1. Понятие и архитектура ODBC;
2. Поименованные источники данных ODBC;
3. Технология «BDE+ODBC»;
4. Поля набора данных. Редактор полей;

5. Иерархия классов полей;
6. Виды полей. Поля синхронного просмотра;
7. Виды полей. Вычисляемые поля;
8. Компонент TDBGrid;
9. Компонент TDBNavigator;
10. Компоненты визуального отображения данных;
11. Компонент TDBLookupComboBox;
12. Компонент TDBCtrlGrid;

Лабораторная работа 3. Язык SQL - базовый интерфейс SQL-сервера.

Цель работы: Изучить основные принципы взаимодействия приложений, разработанных в архитектуре «клиент-сервер», реализовать простейшее клиентское приложение, осуществляющее доступ к базе данных по технологии ODBC (или другой технологии взаимодействия с базами данных), изучить основные принципы работы клиентского приложения с API ODBC и с другими технологиями доступа к базам данных.

Распространение динамически загружаемых библиотек и ресурсов пользовательского интерфейса программ привело к ситуации, когда прикладные программы стали представлять собой не единый программный модуль, а набор взаимосвязанных компонентов. Причем не все компоненты создавались теми же разработчиками, что и сама прикладная программа. Некоторые из них входили в состав ОС, другие поставлялись сторонними разработчиками, которые очень часто могли быть никак не связаны с разработчиками прикладной программы.

При этом любую прикладную программу (приложение) можно условно разделить на четыре основных уровня обработки данных:

- пользовательский интерфейс, отвечающий за отображение данных, представление их пользователю и взаимодействие с ним;
- бизнес-логики, реализующий специализированную логику обработки и преобразования данных, характерную для данной прикладной программы;
- БД, отвечающий за типовые операции получения, хранения, защиты и архивации данных, разделение доступа к ним;
- файловых операций, обеспечивающий работу прикладной программы с файловой системой ОС.

Каждый из этих уровней представляет собой логически цельную составляющую единой прикладной программы. При распределенной обработке данных каждая составляющая может выполняться отдельно (при условии сохранения взаимосвязи между всеми составляющими). В принципе, каждый из этих уровней можно далее разбить на подуровни и получить больше составляющих, общее число которых ничем принципиально не ограничено. Но с другой стороны, все составляющие любой прикладной программы можно разделить всего на две крупные части.

Первая из них обеспечивает нижний уровень работы приложения, отвечает за методы хранения, резервирования, доступа и разделения данных. Вторая организует верхний уровень работы приложения, включает логику обработки данных и интерфейс пользователя. Первая часть обеспечивает два нижних уровня обработки данных. Она, как правило, представляет собой набор компонент, входящих в состав ОС, либо созданных крупными сторонними фирмами-разработчиками, никак не связанными с созданием прикладной программы. Вторая часть обеспечивает реализацию двух верхних уровней обработки данных. Она включает в себя алгоритмы, логику и интерфейс пользователя, созданные разработчиками прикладной программы.

Порядок выполнения работы:

1. Получить вариант задания у преподавателя.
2. Разработать структуру БД в соответствии с полученным заданием, выбрать тип используемой СУБД (по согласованию с преподавателем).

3. Создать БД в соответствии с разработанной структурой. При необходимости реализовать хранимые процедуры, триггеры и другие средства обработки данных на сервере.
4. Изучить принципы построения приложений на основе архитектуры «клиент-сервер». Выбрать технологию для взаимодействия с серверной частью.
5. Выбрать систему программирования для разработки клиентской части приложения (по согласованию с преподавателем).
6. Используя выбранную технологию взаимодействия с серверной частью, с помощью выбранной системы программирования разработать клиентскую часть приложения.
7. Написать и отладить программу на ЭВМ.
8. Подготовить и защитить отчет.
9. Продемонстрировать разработанное приложение преподавателю.

Основные контрольные вопросы:

1. Назовите уровни обработки данных.
2. Назовите основные особенности архитектуры "клиент-сервер"
3. Какие функции может выполнять сервер в архитектуре "клиент-сервер"?
4. Что такое файл-сервер? Назовите особенности его реализации.
5. Какие существуют методы работы с базами данных?
6. Дайте характеристику интерфейса ODBC.
7. Какие источники данных могут быть использованы ODBC?
8. Как осуществляется работа программы при использовании ODBC?

Варианты заданий:

Общие требования к разрабатываемому приложению:

1. Приложение должно реализовывать простейшую прикладную функцию в соответствии с заданием. Конкретный состав выполняемых функций разрабатывается студентами самостоятельно, при необходимости согласовывается с преподавателем.
2. Функции приложения должны выполняться на основе взаимодействия с БД. Состав таблиц и структура БД разрабатывается студентами самостоятельно, согласовывается с преподавателем. Как правило, БД должна содержать 3-5 взаимосвязанных таблиц.
3. Приложение должно работать в двух режимах: административном режиме и режиме пользователя. В административном режиме должно быть доступно больше функций по управлению данными (состав функций для каждого режима указан в варианте задания). Два режима работы приложения должны быть реализованы либо в виде двух программных модулей, либо в виде двух режимов работы одного программного модуля – в этом случае выбор режима работы должен выполняться на основе ввода имени и пароля пользователя при запуске программного модуля.
4. Интерфейс клиентской части приложения должен быть простым, понятным и ориентированным на пользователя. В частности, при выборе данных для взаимосвязанных таблиц выбор должен осуществляться из связанной таблицы с наглядным отображением значащих полей, поиск должен осуществляться по заданным пользователем критериям, при возникновении ошибок, связанных с обработкой данных, должны выдаваться соответствующие сообщения (не сообщения сервера взаимодействия с СУБД!).

Лабораторная работа 4. Физическое проектирование баз данных.

Данная лабораторная работа предназначена для знакомства с основами работы с реляционными базами данных. Условно можно считать, что реляционная база данных состоит из набора таблиц, которые в свою очередь состоят из однотипных записей с несколькими полями. Обычно записи в таблицах представляют в виде горизонтальных рядов, а в колонках - значения соответствующих полей.

Базы данных бывают серверного типа, когда они хранятся в специальном формате, и обращение к ним производится только через специальные программы-серверы баз данных, а также бывают локальные базы. В локальных базах обычно отдельные таблицы хранятся в виде одного или нескольких файлов, при этом базой данных считается каталог на диске, со-

держащий все эти файлы. В нашей работе мы будем работать с локальной базой данных в формате Paradox 7.

В данной работе будет необходимо создать простое приложение для ведения учета книг в домашней библиотеке. Для каждой книги в библиотеке должна быть создана запись в базе данных со следующими полями: фамилия автора, название книги, ее тип и фамилия того, кто взял книгу. Тип книг и список читателей необходимо оформить в виде таблиц-справочников, которые можно будет наполнять по мере необходимости. Просмотр списка книг должен производиться в одном из трех режимов: просмотреть все книги, книги заданного типа, либо книги, находящиеся у выбранного читателя.

В рамках работы необходимо будет освоить программу Database Desktop, предназначенную для создания и редактирования таблиц баз данных. Используя эту программу, нужно будет создать 3 таблицы, которые затем будут подключены к нашей программе в среде Delphi.

Задачи работы

1. Ознакомление с программой Database Desktop и создание в ней 3 таблиц базы данных домашней библиотеки.
2. Краткое знакомство с компонентами для работы с базами данных, находящихся на закладках «Database Access» и «Database Controls».
3. Создание приложения «Домашняя библиотека» для ведения учета книг в домашней библиотеке.

Вопросы и задания для самостоятельной работы

1. Как скрыть поле таблицы от пользователя?
2. Какие бывают типы полей в таблицах? Что означает автоинкрементный тип? Как создаются справочные поля?
3. Как отобразить содержимое таблицы в форме? Какие компоненты нужны для этого?
4. Что такое «курсор» таблицы? Для чего используются в нашей задаче два компонента TTable, связанных с одной физической таблицей? Как устанавливается связь «главный-подчинённый»?
5. Для чего компоненты для работы с базами данными вынесены в отдельный модуль? Можно ли их разместить на главной форме?
6. Как назначить свойство (например DataSource у TDBGrid) указывающим на компонент, находящийся в другой форме?
7. Для чего используются индексы (ключи) в таблицах?
8. Как создать таблицу в Database Desktop?

Лабораторная работа 5. Язык программирования Java.

Цель - изучить практику подготовки и выполнения простых ява-программ.

Краткие теоретические сведения

Java - это один из языков программирования. Он разработан компанией Sun, и является платформо-независимым. Это означает, что программа, написанная на Java, будет одинаково выполнятся и под Windows, и под Linux, и под другими ОС. Достигается это следующим образом - текст программы переводится с помощью компилятора не в родные для процессора команды, а в коды виртуальной java-машины (JVM). Коды виртуальной машины одинаковы на любой платформе, и именно поэтому одна и та же программа и будет работать на разных платформах. Коды эти, кстати, называются байт-кодами. Сама же виртуальная машина от платформы, естественно, зависит - виртуальная машина для Windows отличается от виртуальной машины для других ОС. Мы будем рассматривать создание программ на Java для Windows. Но это не означает, что они не будут работать на других платформах - как раз наоборот.

Существует два основных вида программ на Java - собственно Java-программы и аплеты. Первые выполняются как самостоятельные программы, вторые выполняются в браузере. В настоящее время почти все браузеры имеют в своем распряжении JVM.

Что такое JDK и как его установить?

JDK расшифровывается как Java Developer Kit. Это набор программ и утилит, предназначенный для программирования на Java. В него ряд утилит. Вот некоторые из них:

- Компилятор javac. Именно он и переводит текст программы на Java в байт-коды виртуальной машины.
- Интерпретатор java. Вы с его помощью будете запускать откомпилированные в байт-коды программы. Он содержит в себе JVM (Виртуальную машину Java).
- Утилита appletviewer. С ее помощью можно запускать созданные вами апплеты. Фактически она представляет из себя браузер, который может запускать только апплеты.
- Утилита javadoc. Она предназначена для создания документации.

JDK - это бесплатный набор. Как следствие, с ним придется работать без особого комфорта - тексты программ надо будет набирать в текстовом редакторе. Скачать JDK можно с сайта http://www.oracle.com/us/technologies/java/index.html

После скачивания необходимо инсталлировать пакет на компьютере. По умолчанию установка файлов производится, например для версии jdk1.5.0_08, в "C:\Program Files\Java\jdk1.5.0_08\" и автоматически прописываются "пути" к исполняемому файлу виртуальной ява машины. Для выполнения "компиляции" исходной программы используется компилятор javac.exe, обычно размещаемый "C:\Program Files\Java\jdk1.5.0_08\bin\".

Наиболее простой метод подготовки и выполнения ява программы заключается в следующем:

- На рабочем диске создайте свою инструментальную папку (например m_java - используйте формат имен 8.3, локальные и не буквенные символы не рекомендуются).
- Прейдите в консоль (CMD) и назначите Вашу инструментальную папку текущей.
- Выберите удобный для Вас консольный текстовый редактор. (можно порекомендовать использовать "коммандер консоли"FAR, которым удобно редактировать и просматривать содержимое инструментальной папки. Наиболее удобно работать с "коммандером" в другом окне консоли). Локализация русского языка поддерживается по умолчанию в кодировке 866
- Подготовьте командную процедуру (например j.bat) для консольного вызова компилятора. Конкретный путь с ява компилятору можно определить исследовав папку "C:\Program Files\". Пример пакетного файла приведен ниже: "C:\Program Files\Java\jdk1.5.0_08\bin\javac" %1.java
- Создайте исходный файл, содержащий ява программу(см. ниже). Имя файла и имя класса с методом main должны совпадать. Имена чувствительны к регистру букв.
- Выполните

  j имя_исходного_файла

  java имя_откомпилированного_файла

Первая программа на Java

Первая программа, по давно укоренившейся традиции, будет HelloWorld. Ниже приводится ее текст, который надо набрать в любом текстовом редакторе, позволяющем сохранять документ в ASCII-кодах. Наберите следующий текст и сохраните его в файле HelloWorld.java:

```
class HelloWorld{
   public static void main(String [] args){
      System.out.println("Hello World!");
   }
}
```

Несколько слов по тексту программы. Во-первых, обратите внимание на слово class. Любая программа на Java использует классы. Во-вторых, обязательно должен быть метод (функция) main. Именно с него все и начинается. В нашем случае main имеет несколько модификаторов. Модификатор public означает, что данный метод будет виден снаружи класса. Модификатор static приблизительно означает, что метод main можно вызывать не для кон-

кретного экземпляра класса, а в самом начале программы, когда никакого экземпляра класса еще нет. И, наконец void означает, что метод main не возвращает никакого значения. В строке

```
...
   System.out.println("Hello World!");
...
```

вызывается метод println, который и выводит на экран соответствующую надпись. Этот метод берется из пространства имен System.out.

Еще одно замечание. Java, как и все C-подобные языки, различает строчные и прописные буквы. Так что, например, HelloWorld и helloworld - разные вещи.

После того, как текст набран (напомним, что мы сохраняем его в файле с именем HelloWorld.java), его надо откомпилировать и выполнить.

Задания лабораторной работы.

- подготовьте "рабочее место программиста";
- подготовьте пакетную процедуру компиляции программы;
- выберите и освойте текстовый редактор;
- напишите свое приветствие, содержащее несколько строк текста.

Лабораторная работа 6. Программное обеспечение «среднего» слоя: назначение, архитектура, функции.

Цель работы: Создание элементарного АРМ выполняющего функции сканирования, модификации и хранения информации периферийных датчиков.

Указания к выполнению работы

В общем случае, система АСУ на предприятии включает в себя нескольких уровней:

- уровень контрольно-измерительных приборов и аппаратуры,
- уровень контроллеров (традиционные ПЛК и Softlogic),
- уровень рабочих станций (АРМ).

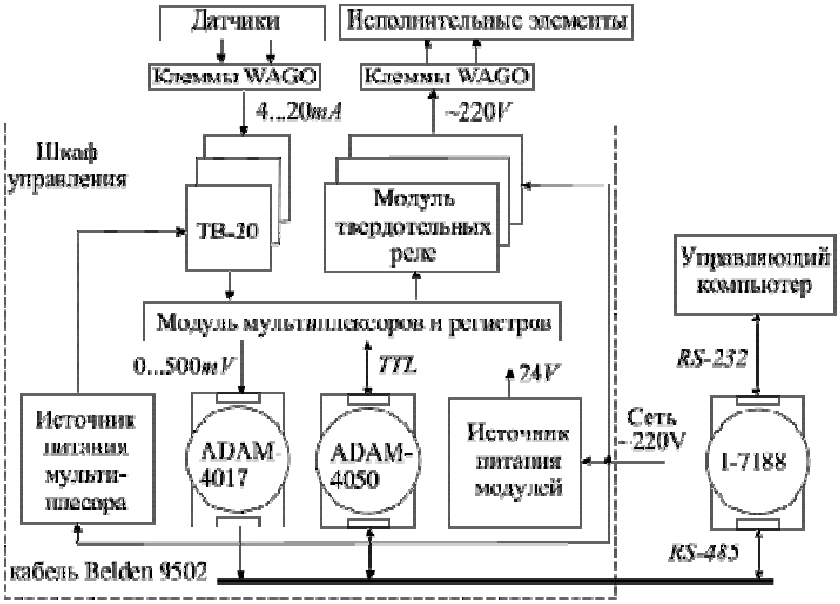
Программное обеспечение (ПО) уровня SCADA выполняет следующие функции;

- Визуализация – отображение информации о процессе на мнемосхемах в виде числовой информации, трендах, анимации, ActiveX компонентов и т. д. Также обеспечивается получение управляющих воздействий от операторов;
- Обмен с контроллерами в реальном времени – обмен информации с контроллерами по последовательному интерфейсу, сети Ethernet;
- Архивирование – сохранение информации;
- Документирование процесса – создание отчётов по заранее созданным шаблонам с последующей их передачей по разным каналам обмена;
- Получение и передача информации в другие программные пакеты – обмен оперативными и архивными данными со специализированным ПО;
- Неоперативная обработка информации – вычисление статистической информации, управление небыстрыми процессами;
- Управление тревогами – список тревог, фильтр событий, квитирование.

Термин Softlogic подразумевает:

Использование контроллеров с PC–совместимой архитектурой. Программирование таких контроллеров осуществляется на языках типа ассемблер или C, а может осуществляться с помощью инструментальных пакетов на визуальных языках верхнего уровня, совместимых со стандартом МЭК61131–3, например, на языке функциональных блоков или языке инструкций.

Структурная схема системы управления может быть представлена как



Управляющие функции выполняет контроллер связи I–7188, который имеет ОЗУ 256 кбайт, флэш-память 512 кбайт, последовательные порты RS–232 и RS–485, порт консоли, сторожевой таймер, часы реального времени и совместим с операционной системой ROM–DOS. По каналу RS–232 контроллер соединен с управляющим компьютером, а по каналуRS–485 (кабель Belden 9502) – со шкафом управления.

В шкафу установлены модули аналогового ввода ADAM–4017 и цифрового ввода–вывода ADAM–4050 фирмы Advantech, модуль мультиплексора и регистров, модули твердотельных реле (Cosmo), источники питания. Для подключения аналоговых сигналов к модулю мультиплексора используются клеммные платы ТВ–20 (Fastwel). Шкаф управления соединен кабелями с датчиками и исполнительными механизмами.

Модуль мультиплексора и регистров позволяет увеличить количество аналоговых и цифровых входов.

Порядок проведения лабораторной работы.

Создайте элементарное АРМ, как указано в разделе «Описание виртуальной лабораторной установки»;

Для задания, предложенного преподавателем, определить объем автоматизации объекта и выбрать комплекс технических средств:

датчики и исполнительные устройства (фирмы Omron, Pepperl+Fuchs, Scaime, Siemens);

средства кабельной разводки и коммутации (фирмы Belden, Hirschmann, Rittal, RST, Schroff, WAGO);

устройства ввода-вывода и связи с объектом – предназначены для ввода сигналов с датчиков в устройство обработки и вывода сигналов для управления исполнительными механизмами(фирмы ADDI-DATA, Advantech, Dataforth, Diamond Systems, Fastwel, Grayhill, LenPromAvtomatika, Lippert, Octagon Systems, Pepperl+Fuchs, Pepperl+Fuchs Elcon, Scaime, Siemens, VIPA, VMIC, WAGO);

рограммируемые контроллеры (фирмы Advantech, Dataforth, Diamond Systems, Fastwel, Grayhill, Lippert, Octagon Systems, Siemens, VIPA, VMIC, WAGO).

Содержание отчета

1. Цель исследований.
2. Итоговый экран создания элементарного АРМ с пояснением функций каждого графического элемента.
3. Задание на автоматизацию элемента участка технологического процесса (по выбору преподавателя).
4. Структурная схема системы управления.

5. Комплекс технических средств реализации структурной схемы с указанием технико-экономических характеристик и функций каждого.
6. Выводы по работе.

Контрольные вопросы и задания

1. Назначение и функции схемы нормализации сигнала.
2. Назначение и функции согласующего устройства.
3. Назначение и функции SCADA систем.
4. Назначение и функции устройства выборки-хранения.

Лабораторная работа 7. Информационные приложения, основанные на использовании "складов данных" (DataWarehousing).

Решение проблемы интеграции неоднородных информационных ресурсов началось с попыток интеграции неоднородных баз данных. Направление интегрированных или федеративных систем неоднородных БД и мульти-БД появилось в связи с необходимостью комплексирования систем БД, основанных на разных моделях данных и управляемых разными СУБД.

Основной задачей интеграции неоднородных БД является предоставление пользователям интегрированной системы глобальной схемы БД, представленной в некоторой модели данных, и автоматическое преобразование операторов манипулирования БД глобального уровня операторы, понятные соответствующим локальным СУБД. В теоретическом плане проблемы преобразования решены, имеются реализации.

При строгой интеграции неоднородных БД локальные системы БД утрачивают свою автономность. После включения локальной БД в федеративную систему все дальнейшие действия с ней, включая администрирование, должны вестись на глобальном уровне. Поскольку пользователи часто не соглашаются утрачивать локальную автономность, желая тем не менее иметь возможность работать со всеми локальными СУБД на одном языке и формулировать запросы с одновременным указанием разных локальных БД, развивается направление мульти-БД. В системах мульти-БД не поддерживается глобальная схема интегрированной БД и применяются специальные способы именования для доступа к объектам локальных БД. Как правило, в таких системах на глобальном уровне допускается только выборка данных. Это позволяет сохранить автономность локальных БД.

Как правило, интегрировать приходится неоднородные БД, распределенные в вычислительной сети. Это в значительной степени усложняет реализацию. Дополнительно к собственным проблемам интеграции приходится решать все проблемы, присущие распределенным СУБД: управление глобальными транзакциями, сетевую оптимизацию запросов и т.д. Очень трудно добиться эффективности. Как правило, для внешнего представления интегрированных и мульти-БД используется (иногда расширенная) реляционная модель данных. В последнее время все чаще предлагается использовать объектно-ориентированные модели, но на практике пока основой является реляционная модель. Поэтому, в частности, включение в интегрированную систему локальной реляционной СУБД существенно проще и эффективнее, чем включение СУБД, основанной на другой модели данных. Основным недостатком систем интеграции неоднородных баз данных является то, что при этом не учитываются "поведенческие" аспекты компонентов прикладной системы. Легко заметить, что даже при наличии развитой интеграционной системы, большинство из указанных выше проблем не решается. Естественным развитием взглядов на информационные ресурсы является их представление в виде набора типизированных объектов, сочетающих возможности сохранения информации (своего состояния) и обработки этой информации (за счет наличия хорошо определенного множества методов, применимых к объекту). Наиболее существенный вклад в создание соответствующей технологии внес международный консорциум OMG, выпустивший ряд документов, в которых специфицируются архитектура и инструментальные средства поддержки распределенных информационных систем, интегрированных на основе общего объектно-ориентированного подхода.

В базовом документе специфицируется эталонная модель архитектуры *(OMA - Object Management Architecture)* распределенной информационной системы.

Согласованная с архитектурой OMA прикладная информационная система представляется как совокупность классов и экземпляров объектов, которые взаимодействуют при поддержке брокера объектных заявок *(ORB - Object Request Broker)*. ORB, общие средства *(Common Facilities)* и объектные службы *(Object Services)* относятся к категории промежуточного программного обеспечения (middleware) и должны поставляться вместе. *Объектные службы* представляют собой набор услуг (интерфейсов и объектов), которые обеспечивают выполнение базовых функций, требуемых для реализации прикладных объектов и объектов категории "общие средства" (например, специфицированы служба именования объектов, служба долговременного хранения объектов, служба управления транзакциями и т.д.). *Общие средства* содержат набор классов и экземпляров объектов, поддерживающих функции, полезные в разных прикладных областях (например, средства поддержки пользовательского интерфейса, средства управления информацией и т.д.).

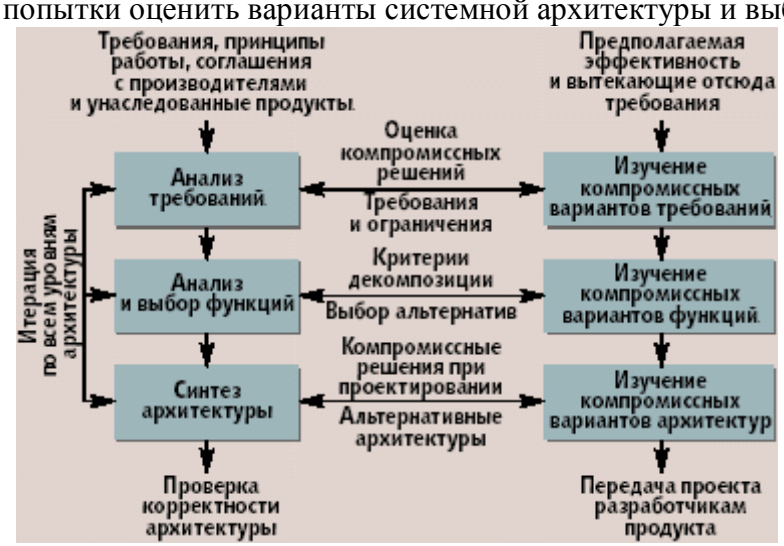
В основе OMA лежит базовая объектная модель *COM (Core Object Model),* в которой специфицированы такие понятия, как объект, операция, тип, подтипизация, наследование, интерфейс. Определены также способы согласованного расширения COM в разных объектных службах.

Интерфейсы объекта-клиента и объекта-сервера должны быть определены на специальном языке *IDL (Interface Definition Language),* который очень напоминает компонент спецификации класса (без реализации) языка Си++. Обращения к ORB могут быть сгенерированы статически при компиляции спецификаций IDL или выполнены динамически с использованием специфицированного в документах OMG API брокера объектных заявок. Правила построения и использования ORB определены в документе OMG *CORBA (Common Object Request Broker Architecture).*

Лабораторная работа 8. Проблема "унаследованных систем" (Legacy Systems).

Системная инженерия определяет систему как совокупность компонентов (физические системы, аппаратное обеспечение, программное обеспечение и персонал), которые совместно реализуют предполагаемые функции продукта. Разработка продуктов – это логическая последовательность оценок затрат на циклы жизни продукта, управление исследованием технических компромиссных решений, анализ эффективности затрат, моделирование эффективности, проектирование архитектуры и процесса изготовления. Все вместе эти шаги преобразуют цели в параметры эффективности и предпочтительную конфигурацию системы.

На рис. 1 показан типичный итеративный процесс системной инженерии. В его рамках выполняются попытки оценить варианты системной архитектуры и выбрать из них наилучшие.



В любой крупной, долгое время существующей корпорации накапливаются информационные подсистемы, разработанные в соответствии с морально устаревшими технологиями. Например, трудно найти корпорацию с возрастом больше 25 лет, в которой не использовались бы информационные подсистемы, созданные на основе ранних аппаратно-программных платформ компании IBM. Базы данных таких подсистем содержат громадные объемы ценной информации, и корпорация просто не может обойтись без их использования. С другой стороны, унаследованные системы очень трудно сопровождать и поддерживать. Очень часто программная часть системы написана на языке ассемблера, а люди, которые писали эти программы, больше не работают в корпорации. Возникают проблемы и с аппаратной частью.

Конечно, для корпорации было бы желательно перевести унаследованные информационные подсистемы на новые технологии, используя, например, какой-либо из подходов, упоминавшихся в нашем курсе. (Заметим, кстати, что применение любого из таких подходов, кроме, быть может, файл-серверных архитектур, практически гарантирует отсутствие проблемы унаследованных систем в будущем. Все эти подходы базируются на концепции открытых систем, на международных или фактически принятых стандартах.) Но беда в том, что работоспособность унаследованной системы может быть настолько важна для корпорации, что эту систему нельзя вывести из использования даже на короткое время.

Одно из наиболее признанных решений проблемы унаследованных систем основывается также на объектно-ориентированном подходе. Идея состоит в том, что "вокруг" системы создается объектная оболочка. Естественно, что при этом порождается и новый объектный интерфейс системы, но до поры сохраняется и ее старый интерфейс. После этого параллельно все другие подсистемы корпорации постепенно переводятся на использование нового интерфейса реконструируемой подсистемы, а сама эта подсистема переделывается в соответствии с современными технологиями. В конце концов, когда сторонние подсистемы полностью готовы работать в новом интерфейсе, и процесс переделки унаследованной подсистемы завершен, она заменяется на вновь разработанный вариант.

В целом идея сравнительно проста, но, конечно, для каждого частного случая приходится решать массу конкретных задач. Это только подход, а не конкретное решение. Но имеется и одна общая задача - научиться унифицированным образом создавать объектные оболочки. Мы не будем явно говорить об этом ниже, но идеи, методы и средства, описываемые в следующих разделах этой части, помогают решить и эту задачу.

4. Перечень используемых программных продуктов

1. Windows XP;
2. MS Office XP;
3. СУБД Microsoft Access;
4. Java;
5. SQL;
6. Язык разметки гипертекста HTML.

5.Фонд тестовых и контрольных заданий

Вопросы к экзамену:

Общее представление об информационной системе.

Общая классификация архитектур информационных приложений.

Средства и методологии проектирования, разработки и сопровождения файл-серверных приложений.

Средства и методологии проектирования, разработки и сопровождения клиент-серверных приложений.

Средства и методологии проектирования, разработки и сопровождения Intranet-приложений.

Информационные приложения, основанные на использовании "складов данных" (DataWarehousing).

Глобально распределенные информационные системы.

Назовите уровни обработки данных.

Назовите основные особенности архитектуры "клиент-сервер"

Какие функции может выполнять сервер в архитектуре "клиент-сервер"?

Что такое файл-сервер? Назовите особенности его реализации.

Какие существуют методы работы с базами данных?

Дайте характеристику интерфейса ODBC.

Какие источники данных могут быть использованы ODBC?

Как осуществляется работа программы при использовании ODBC?

Как скрыть поле таблицы от пользователя?

Какие бывают типы полей в таблицах? Что означает автоинкрементный тип? Как создаются справочные поля?

Как отобразить содержимое таблицы в форме? Какие компоненты нужны для этого?

Что такое «курсор» таблицы? Для чего используются в нашей задаче два компонента TTable, связанных с одной физической таблицей? Как устанавливается связь «главный-подчинённый»?

Для чего компоненты для работы с базами данными вынесены в отдельный модуль? Можно ли их разместить на главной форме?

Как назначить свойство (например DataSource у TDBGrid) указывающим на компонент, находящийся в другой форме?

Для чего используются индексы (ключи) в таблицах?

Как создать таблицу в Database Desktop?

Учебно-методическое обеспечение самостоятельной работы

Карточки с заданиями и методическими указаниями по выполнению практических работ

СТО СМК 4.2.3.05-2011. Стандарт ФГБОУОВПО «АмГУ». Оформление выпускных квалификационных и курсовых работ (проектов), 2011. – 95 с.