Федеральное государственное автономное образовательное
учреждение высшего образования
«Санкт-Петербургский государственный университет
аэрокосмического приборостроения»

Основы программной инженерии

Методические указания к выполнению лабораторных работ

Составители:

к.т.н., стар. преп. П.А. Охтилев, асс. А.Э. Зянчурин

Рецензент:

д.т.н., проф. М.Ю. Охтилев

Санкт-Петербург – 2021

Содержание

Требования к оформлению отчетов

Структура и форма отчета о лабораторной работе

Итоговый отчёт по лабораторным работам должен состоять из частей, перечисленных ниже. Отсутствие указанных ниже частей не допускается. Общий объем отчёта должен составлять не менее 15 страниц. Страницы должны быть пронумерованы. Размер шрифта основного текста — 14 пунктов.

1. *Титульный лист* должен соответствовать образцу на сайте ГУАП. При оформлении титульного листа обязательно наличие следующей информации:
   - название дисциплины;
   - ФИО преподавателя, принимающего работу;
   - ФИО обучающихся, выполнивших работу.
   - Отчёты, содержащие неверную информацию на титульном листе, к сдаче не принимаются.
2. *Содержание* с указанием номеров страниц (желательно составленное автоматически).
3. *Подписанное преподавателем задание* на лабораторные работы.
4. *Краткое описание хода выполнения работ*: постановка задачи на каждую работу; описание использованных моделей, алгоритмов, программных компонент. Описание должно быть сопровождено расчетными материалами, снимками с экрана компьютера («скриншотами»), отражающими ход выполнения работы.
5. *Выводы* по результатам выполняемых лабораторных работ.
6. *Список цитируемой и использованной литературы.*

Требования к оформлению отчета о лабораторной работе

Изложение текста и оформление лабораторных работ следует выполнять в соответствии с ГОСТ 2.105-2019 – ЕСКД и общие требования к текстовым документам ГОСТ 7.32 – 2017 – СИБИД, соблюдая следующие требования.

1. *Оформление титульного листа.* Титульный лист следует оформлять на бланке. Бланки для оформления титульных листов учебных работ представлены на сайте ГУАП в разделе «Нормативная документация» для учебного процесса.
2. *Оформление основного текста работы:*
   - следует использовать шрифт Times New Roman размером не менее 12 пт (допускается 14 пт), строчный, без выделения, с выравниванием по ширине;
   - абзацный отступ должен быть одинаковым и равен по всему тексту 1,25 см;
   - строки разделяются полуторным интервалом;

- поля страницы: верхнее и нижнее – 20 мм, левое – 30 мм, правое – 15 мм;
- полужирный шрифт применяется только для заголовков разделов и подразделов. заголовков структурных элементов;
- разрешается использовать компьютерные возможности акцентирования внимания на определенных терминах, формулах, теоремах, применяя шрифты разной гарнитуры;
- наименования структурных элементов работы: «СОДЕРЖАНИЕ», «ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ», «ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ», «ВВЕДЕНИЕ», «ЗАКЛЮЧЕНИЕ», «СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ», «ПРИЛОЖЕНИЕ» следует располагать в середине строки без точки в конце, прописными (заглавными) буквами, не подчеркивая;
- введение и заключение не нумеруются.
- каждый структурный элемент и каждый раздел основной части следует начинать с новой страницы.

3. *Оформление основной части работы следует делить на разделы и подразделы:*
   - разделы и подразделы должны иметь порядковую нумерацию в пределах всего текста, за исключением приложений;
   - нумеровать их следует арабскими цифрами;
   - номер подраздела должен включать номер раздела и порядковый номер подраздела, разделенные точкой;
   - после номера раздела и подраздела в тексте точка не ставится;
   - разделы и подразделы должны иметь заголовки;
   - если заголовок раздела, подраздела или пункта занимает не одну строку, то каждая следующая строка должна начинаться с начала строки, без абзацного отступа;
   - заголовки разделов и подразделов следует печатать с абзацного отступа с прописной буквы, полужирным шрифтом, без точки в конце, не подчеркивая;
   - если заголовок состоит из двух предложений, их разделяют точкой;
   - переносы слов в заголовках не допускаются;
   - обозначение подразделов следует располагать после абзацного отступа, равного двум знакам относительно обозначения разделов;
   - обозначение пунктов приводят после абзацного отступа, равного четырем знакам относительно обозначения разделов;
   - в содержании должны приводиться наименования структурных элементов, после заголовка каждого из них ставят отточие и приводят номер страницы;
   - содержание должно включать введение, наименование всех разделов и подразделов, заключение, список использованных источников и наименование приложений с указанием номеров страниц, с которых начинаются эти элементы работы;

- перечень сокращений и обозначений следует располагать в алфавитном порядке. Если условных обозначений в отчете менее трех, перечень не составляется.

4. *Оформление нумерации страниц:*
   - страницы следует нумеровать арабскими цифрами, соблюдая сквозную нумерацию по всему тексту работы;
   - номер страницы следует проставлять в центре нижней части листа без точки;
   - титульный лист должен включаться в общую нумерацию страниц;
   - номер страницы на титульном листе не проставляется.
5. *Оформление рисунков:*
   - на все рисунки должны быть ссылки: …в соответствии с рисунком 1;
   - рисунки, за исключением рисунков приложений, следует нумеровать арабскими цифрами;
   - рисунки могут иметь наименование и пояснительные данные, которые помещаются в строке над названием рисунка: Рисунок 1 – Детали прибора
   - рисунки каждого приложения должны обозначаться отдельной нумерацией арабскими цифрами с добавлением перед цифрой обозначения приложения: Рисунок А.3 (третий рисунок приложения А).
6. *Оформление таблиц:*
   - на все таблицы должны быть ссылки, при ссылке следует писать слово «таблица» с указанием ее номера;
   - таблицы, за исключением таблиц приложений, следует нумеровать арабскими цифрами сквозной нумерацией;
   - наименование таблицы следует помещать над таблицей слева, без абзацного отступа: Таблица 1 – Детали прибора
   - таблицы каждого приложения обозначают отдельной нумерацией арабскими цифрами с добавлением перед цифрой обозначения приложения:
     Таблица Б.2 (вторая таблица приложения Б)
   - если таблица переносится на следующую страницу, под заголовком граф должна быть строка с номером колонок, на следующей странице под названием «Продолжение таблицы 1» дается строка с номером колонок.
7. *Оформление приложений:*
   - в тексте отчета на все приложения должны быть ссылки, приложения располагаются в порядке ссылок на них в тексте отчета;
   - каждое приложение следует размещать с новой страницы с указанием в верхней части страницы слова «ПРИЛОЖЕНИЕ»;
   - заголовок приложения записывают с прописной буквы, полужирным шрифтом, отдельной строкой по центру без точки в конце;

- приложения обозначаются прописными буквами кириллического алфавита, начиная с А, за исключением букв Ё, З, Й О, Ч, Ъ, Ы, Ь;
- допускается обозначение приложений буквами латинского алфавита, за исключением I и O;
- в случае полного использования букв кириллического и латинского алфавита допускается обозначать приложения арабскими цифрами;
- приложение следует располагать после списка использованных источников.

8. *Список использованных источников.* Сведения об источниках следует располагать в порядке появления ссылок на источник и в тексте работы и нумеровать арабскими цифрами с точкой и печатать с абзацного отступа. Список использованных источников следует оформлять в соответствии с ГОСТ 7.0.100 – 2018 «Библиографическая запись. Библиографическое описание». Примеры библиографического описания в соответствии с требованиями ГОСТ 7.0.100 – 2018 представлены на сайте ГУАП в разделе «Нормативная документация» для учебного процесса.

Лабораторная работа №1. Разработка требований. Моделирование предметной области. Проектирование программного обеспечения

Цель работы

Целью работы является получение практических навыков, необходимых при формализации требований, моделировании предметной области и проектировании программного обеспечения.

Задание на лабораторную работу

Разработка спецификации требований в соответствии с вариантом задания. Формализация ограничений и бизнес-правил предметной области. Моделирование предметной области и архитектуры программного обеспечения с использованием нотаций UML, BPMN, ER-диаграмм и других языков моделирования по необходимости и согласованию с преподавателем. Выполнение лабораторной работы предполагает коллективное взаимодействие в группе из трех человек в ролях: «Специалист по научно-исследовательским и опытно-конструкторским разработкам», «Системный аналитик», «Архитектор программного обеспечения».

Общие рекомендации по выполнению лабораторной работы

Результатом выполнения лабораторной работы №1 является комплект проектов документов, включаемый в состав отчета по лабораторной работе и отражающий концепцию проекта, модель предметной области и архитектуру программной системы. Комплект документации включает в свой состав следующие документы:

- документ о концепции и границах проекта;
- спецификацию требований к программному обеспечению;
- модель предметной области;
- архитектуру программного обеспечения.

Ниже приведены рекомендации для разработки данных документов.

Документ о концепции и границах проекта

В таблице приведена краткая справка о назначении, характеристиках и наполнении документа.

<table>
<tr><th colspan="2">Документ о концепции и границах проекта</th></tr>
<tr><td colspan="2">Назначение документа</td></tr>
<tr><td colspan="2">Документ о концепции и границах проекта предназначен для формирования первоначального видения о конечном результате работ по проекту, сбора и структуризации первичной информации о предметной области и разрабатываемой программной системе.</td></tr>
<tr><td>Характеристики документа</td><td>Наполнение документа</td></tr>
<tr><td>• однозначно интерпретируемое описание проблемы решаемой в проекте;
• причины возникновения проблемы;
• однозначно интерпретируемое описание практического эффекта от внедрения системы;
• однозначно интерпретируемое описание вариантов (сценариев) использования системы.</td><td>• описание предметной области;
• цель и постановка задачи разработки автоматизированной/информационной системы;
• словарь данных предметной области;
• нежелательные эффекты в предметной области;
• диаграмма вариантов использования;
• дерево функций автоматизированной/информационной системы;
• заключение и выводы.</td></tr>
</table>

Разработчику данного документа важно получить надежное обоснование для реализации проекта. Таким обоснованием должна стать постановка задачи, от которой смогут оттолкнуться все участники проекта. Специалист по научно-исследовательским и опытно-конструкторским разработкам, формирующий цели проекта и видение решения, несет ответственность за реализацию проекта и должен обладать широким кругозором и набором необходимых компетенций по программной инженерии.

Раздел №1. Описание предметной области.

Описание предметной области является одним из важных этапов в формировании понимания «ЧТО» и «КАК» необходимо делать, чтобы достичь поставленных целей при реализации проекта. Ключевым фактором успешного описания предметной области является наличие у специалиста полной и непротиворечивой информации о ней. В момент сбора информации специалист должен выявить *существенные* характеристики объекта автоматизации или анализа и их взаимосвязи.

Алгоритм описания предметной области весьма сложно формализовать, поэтому в рамках данных методических указаний предлагается рассмотреть пример такого описания для наглядности.

Пример. Предметная область «Кафетерий»
Большинство сотрудников предприятия в настоящее время тратят в среднем 65 минут в день, чтобы выбрать, оплатить и съесть обед в кафетерии. Около 20 минут уходит на то, чтобы дойти до кафетерия и вернуться на рабочее место, выбрать еду и оплатить ее наличными или с помощью кредитной карты. Таким образом, сотрудники проводят 90 минут вне рабочих мест. Некоторые звонят в кафетерий заранее, чтобы блюда были готовы к их приходу. Они не всегда получают то, что заказали, потому что некоторые блюда заканчиваются до их прихода. Кафетерий впустую расходует значительный объем продуктов, которые не реализуются, и их приходится выбрасывать. Те же проблемы возникают утром и вечером, хотя гораздо меньше сотрудников завтракает и ужинает, чем обедает.

Раздел №2. Цель и постановка задачи разработки программного обеспечения автоматизированной/информационной системы.

Цель формулируется относительно повышения, например, каких-либо показателей качества, эффективности процессов, сокращения временных издержек или сокращения количества выполняемых операций в процессах предметной области.

В общем случае, постановка задачи определяет содержательные требования и исходные данные для проектирования программной системы. Постановка задачи проектирования кратко отражает ситуацию, требующую изменения, как с точки зрения пользователей, заказчиков и исполнителей, так и с точки зрения предметной области, в которой будет эксплуатироваться программная система. Часто между интересами бизнеса и пользователями существует причинно-следственная связь. К примеру [3]:

Рейтинг удовлетворенности клиентов компании X низок, а доля на рынке уменьшилась на 10% за последний год, потому что у пользователей нет адекватных инструментов, позволяющих посредством решения задач X, Y и Z достичь цели G.

Установление взаимосвязей вопросов исполнителей (разработчиков) с вопросами эксплуатации крайне полезно, когда необходимо убедить заинтересованных лиц в необходимости затрат на проектирование, и позволяет рассматривать этот процесс как с точки зрения пользовательских целей, так и с точки зрения целей исполнителей (разработчиков).

Раздел №3. Словарь данных предметной области.

Формирование словаря данных предполагает идентификацию объектов в исследуемой предметной области, их определение и сопоставление со структурами данных, которые будут использоваться в предполагаемой программной системе.

Простейшие элементы данных

Простейшим элементом данных называется тот, дальнейшая декомпозиция или упрощение которого невозможно или ненужно. К определенным в таблице №1 простейшим элементам данных относятся «Количество контейнеров», «Емкость», «Единица измерения», «Идентификатор заказа» и «Сотрудник, разместивший заказ на химикат». Другие столбцы словаря данных можно использовать для описания относящихся к простейшему элементу данных типу, длине, диапазону числовых значений, списку разрешенных значений (как для «Единицы количества») и других уместных атрибутов [1].

Структура данных

Структура данных (или запись) содержит несколько элементов данных. В таблице №1 показаны следующие структуры данных: «Запрос химиката», «Пункт назначения поставки», «Заказанный химикат» и «Сотрудник, разместивший заказ на химикат». Столбец «Структура или тип данных» в словаре данных – место, где перечисляются элементы, из которых состоит структура, а элементы отделяются знаком «плюс» (+). Структуры могут содержать другие структуры: структура «Сотрудник, разместивший заказ на химикат» включает структуру «Пункт назначения поставки». Каждый элемент данных в структуре должен быть определен в словаре данных [1].

Если элемент в структуре данных необязателен (значение, которое не должно предоставляться пользователем или системой), заключите его в скобки. В структуре «Заказанный химикат» элемент данных «Поставщик» является необязательным, поскольку сотруднику, разместившему запрос, может быть безразлично или он может просто не знать, кто именно поставляет нужный химикат.

Повторяющаяся группа

Если в структуре данных содержится несколько экземпляров элемента данных, заключите этот элемент в фигурные скобки. Покажите допустимое количество возможных повторов в формате «минимум:максимум» перед открывающей скобкой. Например, «Заказанный химикат» в структуре «Заказ химиката» является повторяющейся группой, которая оформляется так: 1:10{Заказанный химикат}. Это означает, что заказ химиката должен содержать как минимум один, но не более десяти химикатов. Если максимальное количество экземпляров в повторяющемся поле неограниченно, для указания этого факта используйте «n». Например, 3:n{повтор} означает,

что определенная структура данных должна содержать не менее трех повторяющихся элементов повтор, а верхней границы на число повторений нет.

Таблица №1

Элемент данных	Описание	Структура или тип данных	Количество	Значение
Заказ химиката	Заказ нового химиката со склада или у поставщика	Идентификатор заказа + Сотрудника, разместивший заказ на химикат + Дата заказа + Счет затрат + 1:10 {Заказанный химикат}		
Пункт назначения	Место куда нужно доставить заказанный химикат			
Количество контейнеров	Количество заказываемых контейнеров определенного размера с химикатом	Положительное целое	3	
Емкость	Объем химиката в заказываемом контейнере	Целое	6	
Единица измерения	Единицы измерения, в которых указывается количество заказываемого химиката	Буквенные символы	10	Граммы
Идентификатор заказа	Уникальный идентификатор	Целое	8	Генерируемый системой порядковый номер, начиная с 1
Заказанный химикат	Описание заказываемого химиката	Идентификатор заказа + Количество контейнеров + Емкость		
Сотрудник, разместивший заказ на химикат	Информация о сотруднике, заказывающем химикат			

Раздел №4. Нежелательные эффекты в предметной области.

Проектирование новых объектов подразумевает улучшение тех или иных технических параметров системы. Сложные задачи требуют особого подхода к их решению, так как улучшение одних параметров системы приводит к недопустимому ухудшению других параметров. Таким образом, возникают противоречия.

Противоречие – взаимодействие противоположных, взаимоисключающих сторон и тенденций предметов и явлений, которые вместе с тем находятся во внутреннем единстве и взаимопроникновении, выступая источником самодвижения и развития объективного мира и познания [5].

Противоречия выражаются в виде нежелательных эффектов. Нежелательный эффект это некая характеристика в предметной области, которую необходимо устранить или изменить ее состояние для улучшения функционирования системы в целом.

Выявление нежелательного эффекта реализуется выявлением конфликтующих элементов системы. Для их выявления следует построить таблицу взаимодействия, которая имеет следующий вид [5]:

Таблица №2

Элементы	**Элемент$_1$**	**Элемент$_2$**	**Элемент$_3$**	**…**	**Элемент$_n$**
Элемент$_1$		+/-	+/-	+/-	+/-
Элемент$_2$			+/-	+/-	+/-
Элемент$_3$				+/-	+/-
…					+/-
Элемент$_n$					

Примечание. В таблице №2 обозначено:

- «+» – наличие конфликта (нежелательный эффект);
- «-» – отсутствие конфликта;
- ▒▒▒ – связь не рассматривается.

Раздел №5. Диаграмма вариантов использования.

При выполнении анализа информации о предметной области необходимо сопоставлять различные ее представления на предмет обнаружения пробелов, ошибок и противоречий.

Один из точных способов поиска недостающих элементов системы для понимания – матрица *CRUD* (*Create, Read, Update, Delete* – создание, чтение, обновление, удаление). Она позволяет соотнести действия предполагаемой системы с элементами данных, что дает более конкретное представление о

том, где как каждый элемент создается, считывается, обновляется и удаляется. Иногда добавляют к названию матрицы букву L указывая, что элемент данных является списком (*list*), или *M* или *C* – соответственно для обозначения перемещения или копирования данных из одного места в другое [1].

В таблице №3 показан пример матрицы *CRUD* Каждая ячейка указывает, как вариант использования, определенный в крайнем левом столбце, использует элементы данных, показанные в остальных столбцах. Вариант использования может создать (*C*), прочитать (*R*), обновить (*U*) или удалить (*D*) сущность. После создания матрицы посмотрите, нет ли ячейки столбца, в которой нет одной из этих букв. Например, если сущность обновлена, но до этого ее не создавали, то откуда она взялась?

Таблица №3

Сущность / Варианты использования	**Заказ**	**Химикат**	**Сотрудник, разместивший заказ на химикат**	**Каталог поставщика**
Разместить заказ	*C*	*R*	*R*	*R*
Изменить заказ	*U, D*		*R*	*R*
Управлять складом химикатов		*C,U,D*		
Создать отчет по заказу	*R*	*R*	*R*	
Редактировать сотрудников, имеющих право размещать заказы			*C,U*	

В таблице №3 показано, что ни одна ячейка в столбце «Сотрудник, разместивший заказ на химикат» не содержит *D*. То есть ни в одном из случаев использования нельзя удалить сотрудника из списка людей, заказывавших химикаты. Интерпретировать это можно тремя способами [1]:

- удаление сотрудника, разместившего заказ на химикат, не является ожидаемой функцией программной системы;
- не был учтен вариант использования, который удаляет сотрудника, разместившего заказ на химикат;
- вариант использования «Редактировать сотрудников, имеющих право размещать заказы» некорректный. Предполагается, что пользователь может удалить из списка сотрудника, размещающего заказ на химикат, но в настоящее время эта функциональность не указана в вариантах использования.

В данном случае невозможно установить, какая интерпретация правильна, но *CRUD* – это надежный способ для обнаружения недостающих требований.

Раздел №6. Дерево функций программного обеспечения автоматизированной/информационной системы.

Дерево функций программной системы автоматизированной/информационной системы следует начать с определения основных ее основных функций в формате списка ниже [1, 7].

1. Функция №1.
2. Функция №2.
3. Функция №3.
4. …

Затем, построить дерево функций, которое имеет следующий вид.

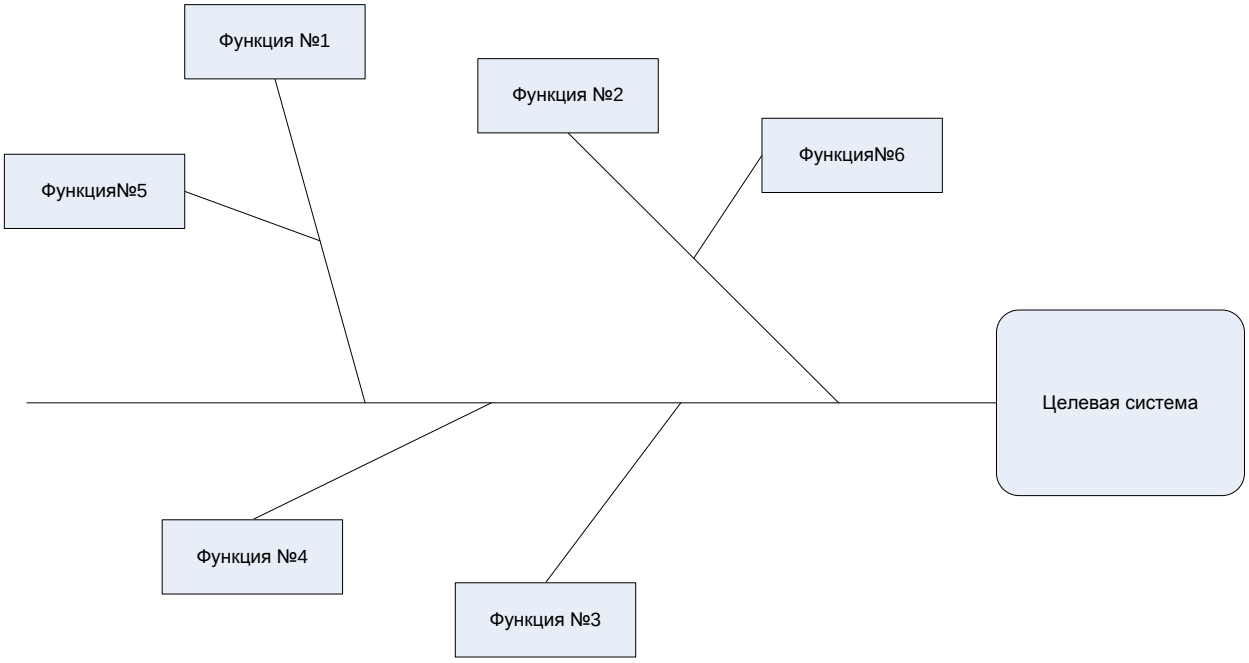

Рис. 1. Пример дерева функций программной системы

Построение дерева функций следующий состоит из следующих шагов.

1. Построить основную линию, ведущую к блоку «Целевая система» (программная система).
2. Построить блоки ассоциированные с основными функциями программной системы и построить «ветки» к основной линии (на рис. 1 «Функция №1», «Функция №2», «Функция №3», «Функция №4»).
3. Построить блоки ассоциированные с второстепенными функциями, но связанные с основными (на рис. 1. «Функция№5» второстепенная, но связанная с «Функция№1»).

Функции должны отражать вариант использования программной системы и/или должен быть направлен на устранение нежелательных эффектов, возникающих в предметной области.

Раздел №7. Заключение и выводы.

В данном разделе необходимо привести пронумерованный перечень выводов, которые были сделаны в результате анализа предметной области.

Спецификация требований к программному обеспечению

В таблице приведена краткая справка о назначении, характеристиках и наполнении документа.

Спецификация требований к программному обеспечению	
Назначение документа	
Спецификация требований к программному обеспечению предназначена для формализации и фиксации бизнес-требований, функциональных и нефункциональных требований, требований к пользовательским интерфейсам.	
Характеристики документа	**Наполнение документа**
• обеспечение полноты требований к системе в целом; • обеспечение непротиворечивости требований; • обеспечение согласованности требований; • обеспечение корректности требований; • обеспечение верифицируемости требований; • обеспечение упорядоченности требований по важности и стабильности.	• определение бизнес-требований; • определение функциональных требований; • определение нефункциональных требований; • заключение и выводы.

Раздел №1. Определение бизнес-требований

Бизнес-требования это требования к системе в целом (системные требования). Указываются требования к [1]:

- структуре системы (здесь закладываются высокоуровневые архитектурные решения, либо структурные ограничения, вводится деление на подсистемы, комплексы и модули, решаются вопросы коммуникации компонент системы и системы с внешним миром);
- режимам функционирования системы;
- персоналу (указывается численность, требуемая квалификация и режим работы);
- надежности;
- безопасности;
- эргономике и технической эстетике;
- транспортабельности для подвижных автоматизированных систем;
- эксплуатации, техническому обслуживанию, ремонту и хранению компонентов системы;
- защите информации от несанкционированного доступа;
- сохранности информации при авариях;

- защите от влияния внешних воздействий;
- патентной чистоте;
- стандартизации и унификации, а также показатели назначения (параметры, характеризующие степень соответствия системы ее назначению) и дополнительные требования (распространяются на обучающие подсистемы, средства контроля работоспособности системы и др.).

Раздел №2. Определение функциональных требований

При составлении функциональных требований следует учитывать следующее [7]:

- ограничения предметной области;
- внешние интерфейсы автоматизированной/информационной системы.

Функциональные требования – это операции или действия, которые должны выполняться с объектами предметной области и которые, как правило, ассоциированы с интерфейсными элементами управления. Функциональные требования можно считать действиями программной системы. Кроме того, функциональные требования определяют места или контейнеры, с помощью которых объекты или данные отображаются пользователю. Функциональные требования определяют функциональность (поведение) программной системы, которая должна быть создана разработчиками для предоставления возможности выполнения пользователями своих обязанностей в рамках бизнес-требований и в контексте пользовательских требований.

Раздел №3. Определение нефункциональных требований

Ограничения

Формулировки условий, модифицирующих требования или наборы требований, сужая выбор возможных решений по их реализации. В частности, к ним могут относиться параметры производительности, влияющие на выбор платформы реализации и/или развертывания (протоколы, серверы приложений, БД, ...), которые, в свою очередь, могут относиться, например, к внешним интерфейсам.

Внешние интерфейсы

Конкретизация аспектов взаимодействия с другими системами, операционной средой (например, запись в журнал событий операционной системы), возможностями мониторинга при эксплуатации и т.д.

Атрибуты качества

Описывают дополнительные характеристики продукта в различных “измерениях”, важных для пользователей и/или разработчиков. Атрибуты

касаются вопросов портируемости, прозрачности взаимодействия с другими системами, целостности, устойчивости и т.п.

Информационные требования, предъявляемые к пользовательским интерфейсам

Потребности пользователя в данных – это объекты и информация, которые должна предоставлять система. Если говорить в терминах приведенной выше семантики, бывает полезно считать информационные требования объектами и прилагательными, связанными с этими объектами. Например, учетные записи, люди, документы, сообщения, песни, изображения, а также их свойства, такие как состояние, дата, размер, автор, тема [3].

Функциональные требования, предъявляемые к пользовательским интерфейсам

Принципы разработки функциональных требований, предъявляемые к пользовательским интерфейсам аналогичны приведенным в Разделе №2.

Раздел №4. Заключение и выводы

В данном разделе необходимо привести пронумерованный перечень выводов, которые были сделаны в результате определения всех видов требований.

Модель предметной области

В таблице приведена краткая справка о назначении, характеристиках и наполнении документа.

Модель предметной области	
Назначение документа	
Документ предназначен для структурированного описания объектов и процессов предметной области с использованием нотаций языков моделирования.	
Характеристики документа	**Наполнение документа**
• обеспечение согласованности со спецификацией требований; • обеспечение корректности разрабатываемых моделей как с точки зрения языка моделирования, так и с точки зрения ; • обеспечение полноты при моделировании предметной области.	• диаграмма прецедентов; • диаграмма объектов; • диаграмма бизнес-процессов; • диаграмма состояний; • заключение и выводы.

Раздел №1. Диаграмма прецедентов

Программные системы проектируются с учетом того, что в процессе работы они будут использоваться людьми и/или взаимодействовать с другими системами. Сущности, с которыми взаимодействует система в процессе работы, называются *акторами*, причем каждый *актор* ожидает, что система будет вести себя строго определенным, предсказуемым образом.

Актор (actor) – это множество логически связанных ролей, исполняемых при взаимодействии с прецедентами или сущностями (система, подсистема или класс). *Актором* может быть пользователь или другая система, подсистема или класс, которые представляют нечто вне сущности.

Пример графического изображения роли *актор* представлен на рисунке 2 в двух вариантах. Обе формы представления имеют один и тот же смысл и могут использоваться в диаграммах. "Стереотипированная" форма чаще применяется для представления системных *акторов* или в случаях, когда *актор* имеет свойства и их нужно отобразить [6].

Рис. 2. Пример вариантов графического изображения роли «актор»

Также центральным понятием в данных типах диаграмм является понятие «прецедент».

Прецедент (use-case) – описание отдельного аспекта поведения системы с точки зрения пользователя [6].

Прецедент (use-case) – описание множества последовательных событий (включая варианты), выполняемых системой, которые приводят к наблюдаемому *актором* результату. Прецедент представляет поведение сущности, описывая взаимодействие между *акторами* и системой. Прецедент не показывает, "КАК" достигается некоторый результат, а только "ЧТО" именно выполняется.

Прецеденты обозначаются очень простым образом – в виде эллипса, внутри которого указано его название. *Прецеденты и экторы соединяются с помощью линий*. Часто на одном из концов линии изображают *стрелку*, причем *направлена она к тому, у кого запрашивают сервис*, другими словами, чьими услугами пользуются. Это простое объяснение иллюстрирует *понимание прецедентов как сервисов*, пропагандируемое компанией IBM.

Рис. 3. Пример графического изображения элемента «перцедент»

Прецеденты могут включать другие *прецеденты*, расширяться и наследоваться ими.

Пример диаграммы прецедентов представлен на рисунке 4.

Рис. 4. Пример диаграммы прецедентов в предметной области «библиотека»

В данном примере изображены два *актора* «библиотекарь» и «студент», а также какие действия, как ожидается, они будут выполнять в условной предметной области «библиотека». В качестве комментария приведен элемент «примечание», уточняющий, что понимается под прецедентом «Консультации». Следует заметить, что иногда на диаграммах прецедентов *границы системы обозначают прямоугольником*, в верхней части которого может быть указано *название предметной области или программной системы*. Таким образом, прецеденты – действия, выполняемые впредметной области в ответ на действия *актора*.

Для большего понимания ниже приводится еще один пример диаграммы прецедентов предметной области «универсистет» на рисунке 5.

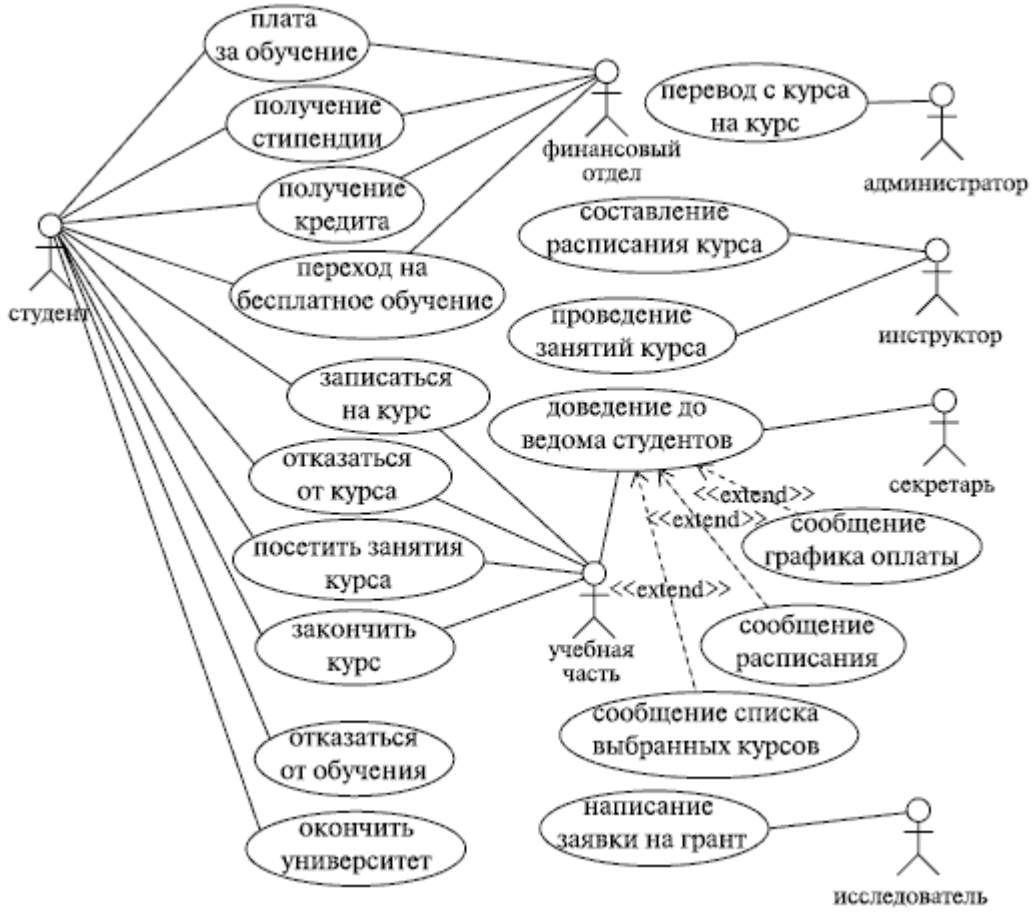

Рис. 5. Пример диаграммы прецедентов предметной области «университет»

Таким образом, можно сделать вывод о том, что *диаграммы прецедентов* относятся к той группе диаграмм, которые представляют динамические или поведенческие аспекты системы. Данное представление служит *средством для достижения взаимопонимания между разработчиками, экспертами и конечными пользователями* программной системы. Характерная черта таких диаграмм заключается в простом и наглядном для понимания и восприятия представлении, что существенно сокращает трудозатраты на согласование вопросов при моделировании предметной области.

Подводя итог, данного раздела можно выделить такие *цели создания диаграмм прецедентов*:

- определение границы и контекста моделируемой предметной области на ранних этапах проектирования;
- формирование общих требований к поведению проектируемой системы;
- разработка концептуальной модели системы для ее последующей детализации;
- подготовка документации для взаимодействия с заказчиками и пользователями системы.

Раздел №2. Диаграмма объектов

В диаграмме объектов центральным понятием является «объект». Дадим ряд определений данному понятию.

1. *Объект (object)* – экземпляр класса.
2. *Объект (object)* – конкретная материализация абстракции;
3. *Объект (object)* – сущность с хорошо определенными границами, в которой инкапсулированы состояние и поведение;
4. *Объект (object)* – экземпляр класса (вернее, классификатора – *актор*, класс или интерфейс). Объект уникально идентифицируется значениями атрибутов, определяющими его состояние в данный момент времени.

В рамках данных методических указаний будем руководствоваться понятием №4.

Для большего понимания рассмотрим следующий пример – все мы (люди) являемся объектами класса "человек" и различимы между собой по таким признакам (значениям атрибутов), как имя, цвет волос, глаз, рост, вес, возраст и т. д. (в зависимости от того, какую задачу мы рассматриваем и какие свойства человека для нас в ней важны).

Пример графического изображения элемента диаграммы «объект» представлен на рисунке 6. Под словом имя здесь мы понимаем название объекта и наименование его класса, разделенные двоеточием. Для указания значений атрибутов объекта в его обозначении может быть предусмотрена специальная секция. Еще один нюанс состоит в том, что объект может быть анонимным: это нужно в том случае, если в данный момент не важно, какой именно объект данного класса принимает участие во взаимодействии [6].

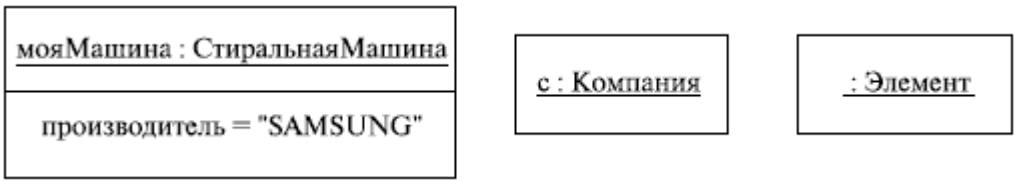

Рис. 6. Пример графического изображения элемента диаграммы «объект»

Диаграммы объектов демонстрируют множество объектов – экземпляров классов и отношений между ними в некоторый момент времени. Другими словами, *диаграмма объектов – это своего рода снимок состояния системы в определенный момент времени*, показывающий множество объектов, их состояния и отношения между ними в данный момент.

Таким образом, *диаграммы объектов* представляют *статический вид системы с точки зрения проектирования и процессов*, являясь основой для сценариев, описываемых диаграммами действий, о которых речь пойдет в следующем разделе.

Приведем пример диаграммы объектов некоторого предприятия в ситуации продвижения нового товара или услуги.

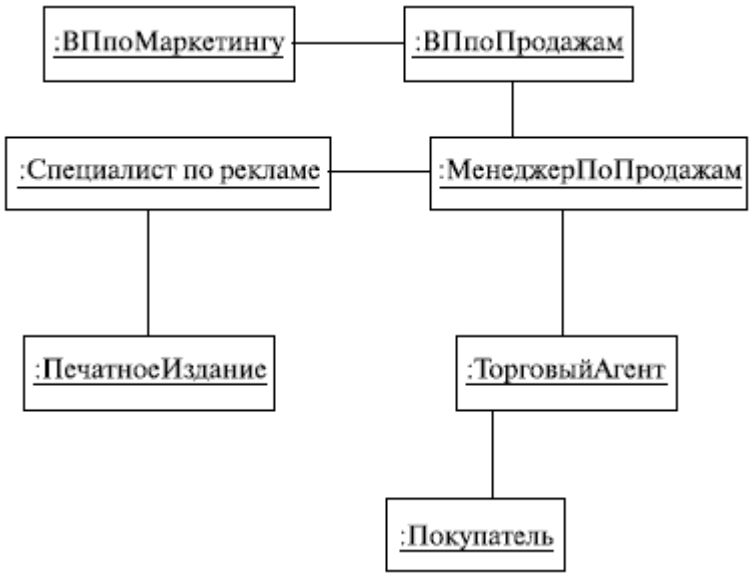

Рис. 7. Пример диаграммы объектов предприятия при продвижении новых товаров или услуг

В данном примере наглядно демонстрируется какие должностные лица с кем взаимодействуют. В данных типах диаграмм не отражается как и какой информацией обмениваются вовлеченные в процесс участники, зато наглядно видна структура взаимосвязей без конкретного именования представленных участников.

На рисунке 8 проиллюстрирован пример типовой организационно-штатной структуры предприятия.

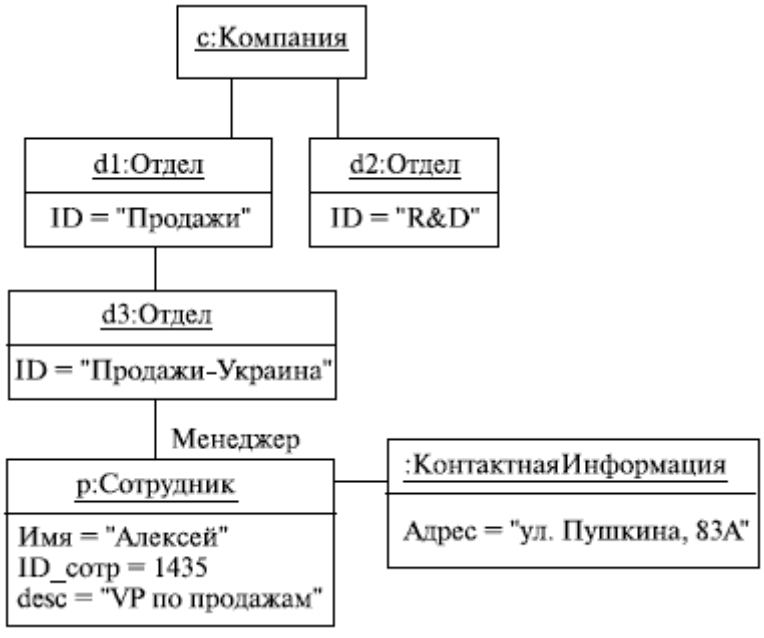

Рис. 8. Пример диаграммы объектов типовой организационно-штатной структуры предприятия

Раздел №3. Диаграмма бизнес-процессов

Business Process Model and Notation (BPMN) – международный стандарт моделирования, разработанный организацией BPMI. Основной целью языка BPMN является обеспечение абсолютно доступной нотацией для описания бизнес-процессов всех бизнес-пользователей: от бизнес-аналитиков, создающих схемы процессов, и разработчиков, ответственных за внедрение технологий выполнения бизнес процессов, до руководителей и обычных пользователей, управляющих этими бизнес-процессами и отслеживающих их выполнение. Таким образом, BPMN нацелен на устранение расхождения между моделями бизнес-процессов и их реализацией.

Описание элементов BPMN приведено в на рисунке 9.

Общее представление

За короткий промежуток времени был сделан большой скачок в разработке языков на основе XML для реализации бизнес-процессов для Web-сервисов. Такие языки, как WSBPEL (Web Services Business Process Execution Language), служат для формального описания бизнес-процессов. Отличительной особенностью языков такого формата является то, что они были оптимизированы для описания процессов внутри BPM-систем и их взаимодействия с другими. Оптимизация этих языков для приложений сделала их менее удобными для использования людьми (включая разработку бизнес-процессов, управление ими, мониторинг). Для описания бизнес-процессов WSBPEL использует блоки и диаграммы. В нем применены принципы формальных математических моделей (например, pi-calculus1). Все это способствовало формированию основ выполнения бизнес-процессов для обработки комплексных внутренних и внешних B2B взаимоотношений, а также выгодного использования Web-сервисов. Для WSBPEL характерно то, что сложные бизнес-процессы могут быть организованы в формате потенциально сложных, разрозненных и интуитивно непонятных моделей, которые с легкостью обрабатываются программами и приложениями, однако, сложны для понимания системными-аналитиками и руководителями проектов, задачами которых являются разработка, управление и мониторинг процессов. Языки на основе XML для реализации бизнес-процессов для Web-сервисов не ориентированы на удобство использования и восприятия людей не связанных с областью программирования [2].

Область применения

В данных методических указаниях рассматриваются нотация, варианты моделирования бизнес-процессов, а также формат обмена данными, применение которых поможет пользователям успешно осуществлять использование терминов нотации BPMN (как в моделях, так и на диаграммах) и их замещение при работе с различными инструментами моделирования. Задача данного раздела документа – показать гибкость использования одних и

Рис. 9. Основные элементы нотации моделирования бизнес-процессов (BPMN)

тех же объектов нотации в различных бизнес-процессов рассматриваемой предметной области.

BPMN заключает в себе концепцию моделирования, применяемую для бизнес-процессов, что, соответственно, исключает рассмотрение других типов моделирования, осуществляемого различными организациями для ведения деловой деятельности. По этой причине в данном разделе не концентрироваться на следующих аспектах [2]:

- определение типов организационных структур и их ресурсов;
- функциональное моделирование структуры организации;
- создание моделей данных и информационных моделей;
- моделирование стратегии;
- создание бизнес-правил.

Задача состоит в том, чтобы формализовать взаимодействие между огромным количеством информации и множеством целевых групп, выявленных в предметной области.

Раздел №4. Диаграмма состояний

Объекты характеризуются поведением и состоянием, в котором находятся. *Диаграммы состояний применяются для того, чтобы объяснить, каким образом работают сложные объекты*. Приведем определение понятия «состояние».

Состояние (state) – ситуация в жизненном цикле объекта, во время которой он удовлетворяет некоторому условию, выполняет определенную деятельность или ожидает какого-то события. Состояние объекта определяется значениями некоторых его атрибутов и присутствием или отсутствием связей с другими объектами.

Диаграмма состояний показывает, как объект переходит из одного состояния в другое. Очевидно, что диаграммы состояний служат для моделирования динамических аспектов системы (как и диаграммы последовательностей, кооперации, прецедентов и, как мы увидим далее, диаграммы деятельности). Диаграмма состояний полезна при моделировании жизненного цикла объекта (как и ее частная разновидность – диаграмма деятельности, о которая не рассматривается в данных методических указаниях).

От других диаграмм диаграмма состояний отличается тем, что описывает процесс изменения состояний только одного экземпляра определенного класса – одного объекта, причем объекта *реактивного*, то есть объекта, поведение которого характеризуется его реакцией на внешние события. Понятие жизненного цикла применимо как раз к реактивным объектам, настоящее состояние (и поведение) которых обусловлено их прошлым состоянием. Но диаграммы состояний важны не только для

описания динамики отдельного объекта. Они могут использоваться для *конструирования исполняемых систем* путем прямого и обратного проектирования [6].

Далее приведем описание диаграммы состояний (рис 10). Скругленные прямоугольники представляют состояния, через которые проходит объект в течение своего жизненного цикла. Стрелками показываются переходы между состояниями, которые вызваны выполнением методов описываемого диаграммой объекта. Существует также *два вида псевдосостояний: начальное*, в котором находится объект сразу после его создания (обозначается сплошным кружком), и *конечное*, которое объект не может покинуть, если перешел в него (обозначается кружком, обведенным окружностью).

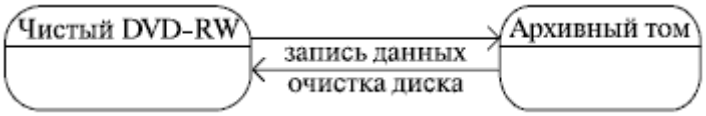

Рис. 10. Пример простой диаграммы состояний

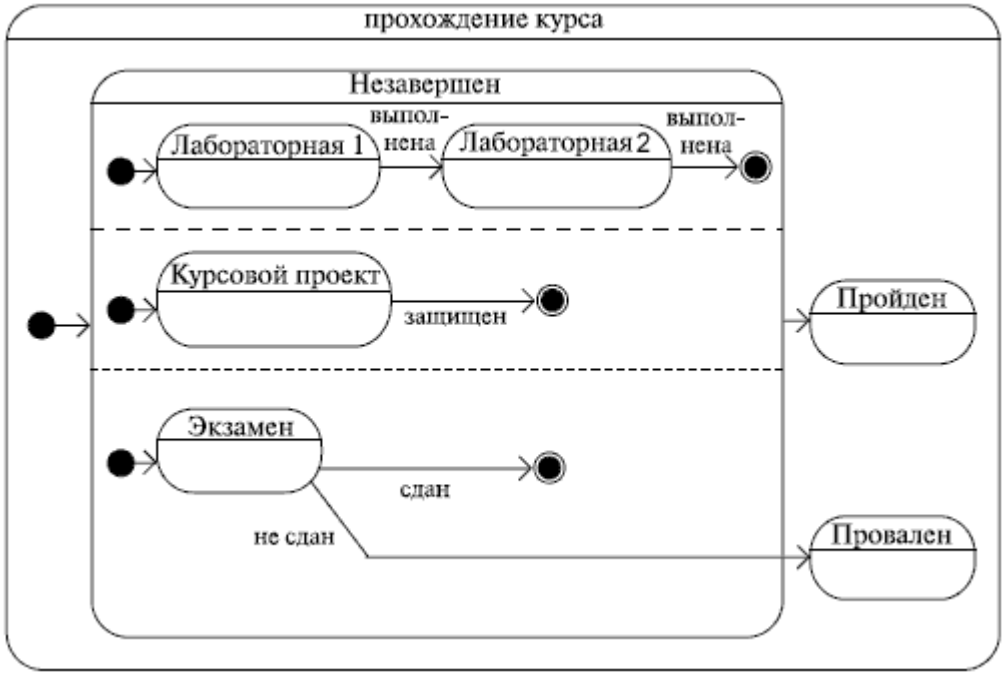

Рис. 11. Пример составной диаграммы состояний на примере «прохождение курса»

В примере «прохождение курса» на рисунке 11 можно заметить, что составное состояние, включающее другие состояния, одно из которых содержит также параллельные подсостояния. Суть процесса заключается в следующем. Для того чтобы пройти курс, студент должен выполнить лабораторные работы, защитить курсовой проект и сдать экзамен.

Для лучшего понимания предлагается рассмотреть еще один пример, приведенный на рисунке 12.

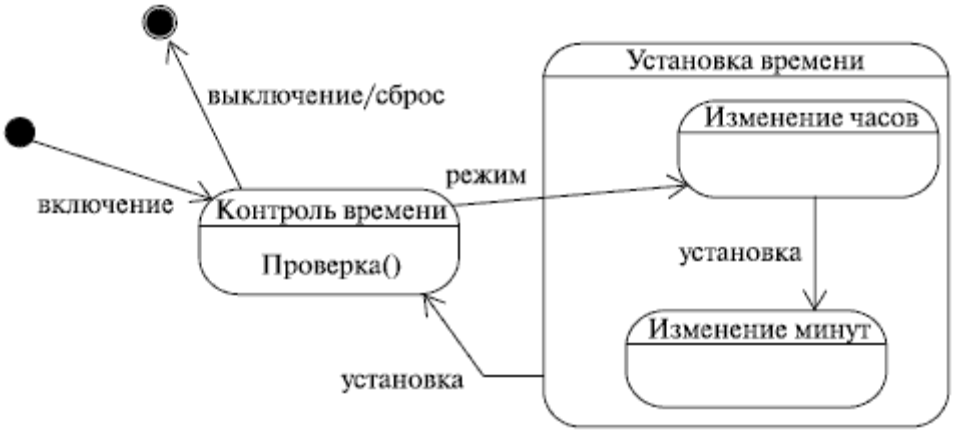

Рис. 12. Пример диаграммы состояний «Таймер»

Основное его назначение таймера – контроль времени (проверка, не истек ли указанный промежуток), но у него есть еще один режим работы – установка. По истечении указанного промежутка времени или при "сбросе" устройство отключается.

Раздел №5. Заключение и выводы

В данном разделе необходимо привести пронумерованный перечень выводов, которые были сделаны в результате формализации объектов и процессов предметной области.

Архитектура программного обеспечения

В таблице приведена краткая справка о назначении, характеристиках и наполнении документа.

Архитектура программного обеспечения	
Назначение документа	
Документ предназначен для структуризации блоков кода (модулей) системы, выявления их поведенческих особенностей и интерфейсов взаимодействия.	
Характеристики документа	**Наполнение документа**
• обеспечение согласованности с требованиями и моделями предметной области; • обеспечение непротиворечивости с требованиями и моделями предметной области; • обеспечение связности модулей друг с другом; • обеспечение корректности описания архитектурных структур; • обоснование выбора архитектуры.	• определение состава структурных элементов программного обеспечения автоматизированной информационной системы, их назначения и интерфейсов; • определение варианта архитектуры и ее разработка; • заключение и выводы.

Раздел №1. Определение состава структурных элементов архитектуры программного обеспечения автоматизированной/информационной системы, их назначения и интерфейсов

Современные программные системы на сегодняшний день сложны для восприятия при их комплексном рассмотрении, поэтому принято выделять структурные элементы программной системы и концентрироваться на их модулях и компонентах.

Для определения архитектуры программного обеспечения предлагается рассмотреть следующую классификацию представлений архитектурных структур [4].

1. *Модульные структуры*. Элементами таких структур являются модули – блоки реализации. Модули предполагают рассмотрение системы с точки зрения программного кода. Модульные структуры позволяют отвечать на такие вопросы, как: «Какие функциональные требования выполняет данный модуль?», «К каким программным элементам он может обращаться?», «Какое программное обеспечение он фактически использует?», «Между какими модулями установлены отношения обобщения или специализации (например, наследования)?».
2. *Структуры компонент соединитель*. В данном случае элементами являются компоненты (основные единицы вычислений) и соединители (инструменты взаимодействия между компонентами) периода прогона

(выполнения). Среди вопросов, на которые отвечают структуры «компонент» и «соединитель», такие, например, как: «Каковы основные исполнительные компоненты и как происходит их взаимодействие?», «Каковы основные совместно используемые хранилища данных?», «Какие части системы воспроизводятся?», «Каким образом по системе проходят данные?», «Какие элементы системы способны выполняться одновременно?», «Какие структурные изменения происходят в системе во время ее исполнения?».

3. *Структуры распределения*. Структуры распределения демонстрируют связь между программными элементами, с одной стороны, и элементами одной или нескольких внешних сред, в которых данное программное обеспечение создается и исполняется – с другой. Они отвечают на вопросы: «На каком процессоре исполняется данный программный элемент?», «В каких файлах каждый элемент хранится в ходе разработки, тестирования и конструирования системы?», «Каким образом программные элементы распределяются между группами разработчиков?».

Рассмотрим подробнее каждый из классов архитектурных структур.

Модульные структуры

1. *Декомпозиция*. В качестве блоков выступают модули, между которыми установлены отношения “является подмодулем...”. Таким образом, крупные модули в рекурсивном порядке разлагаются на меньшие, и этот процесс завершается только тогда, когда меньшие модули становятся вполне понятными. Модули в рамках этой структуры часто используются в качестве отправной точки для последующего проектирования – архитектор перечисляет блоки, с которыми ему предстоит работать, и в расчете на более подробное проектирование, а также реализацию, распределяет их между модулями. Структура декомпозиции в значительной степени обеспечивает модифицируемость системы – при этом складывается ситуация, когда наиболее вероятные изменения приходятся на долю нескольких небольших модулей.
2. *Варианты использования*. Блоками этой важной, но не слишком распространенной структуры могут быть либо модули, либо (когда требуется более мелкая структура) процедуры или ресурсы интерфейсов модулей. Между такими блоками устанавливаются отношения использования. Структура использования полезна при конструировании систем, которые легко расширяются дополнительными функциями, либо предлагают возможность быстрого извлечения полезных функциональных подмножеств. Способность без труда разбить рабочую систему на ряд подмножеств подразумевает возможность инкрементной разработки – многофункционального конструкторского приема.
3. *Многоуровневые*. Если отношения использования в рамках этой структуры находятся под строгим, особым образом осуществляемом

контроле, возникает система уровней – внутренне связанных наборов родственных функций. В рамках строгой многоуровневой структуры уровень N может обращаться к услугам только в том случае, если они представлены уровнем N-1. На практике это правило существует в виде многочисленных вариантов, которые частично снимают приведенное структурное ограничение. Уровни во многих случаях проектируются в виде абстракций (виртуальных машин), которые, стараясь обеспечить переносимость, скрывают детали своей реализации нижележащих уровней от вышележащих.

4. *Класс или обобщение*. Блоки модулей в рамках этой структуры называются классами. Отношения между ними строятся по образцам “наследуют от...” и “являются экземпляром...”. Данное представление способствует анализу коллекций сходного поведения или сходных возможностей (например, классов, которые наследуют от других классов) и параметрических различий, фиксация которых производится путем определения подклассов. Структура классов позволяет анализировать вопросы повторного использования и инкрементного введения функциональности.

Структуры «компонент» и «соединитель»

1. *Процесс или сообщающиеся процессы*. Подобно другим структурам “компонент и соединитель”, эта является ортогональной по отношению к модульным структурам, поскольку она связана с динамическими аспектами исполняемой системы. В качестве блоков в данном случае выступают процессы или потоки, связь между которыми устанавливается путем передачи данных, синхронизации и/или операций исключения. Здесь, как и во всех остальных структурах “компонент и соединитель”, действует отношение прикрепления, демонстрирующее связь компонентов друг с другом. Структура процессов существенно облегчает конструирование рабочей производительности и готовности системы.
2. *Параллелизм*. Данная структура позволяет архитекторам выявлять перспективы параллелизма и локализовать возможности состязаний за ресурсы. В качестве блоков выступают компоненты, а соединители играют роль “логических потоков”. Логическим потоком называется такая последовательность вычислений, которую впоследствии, в ходе процесса проектирования, можно связать с отдельным физическим потоком. Структура параллелизма задействуется на ранних этапах проектирования и способствует выявлению требований к организации параллельного исполнения.
3. *Совместно используемые данные или репозиторий*. В состав данной структуры входят компоненты и соединители, обеспечивающие создание, хранение и обращение к данным постоянного хранения. Она наилучшим образом приспособлена к таким ситуациям, когда система

структурирована на основе одного или нескольких репозиториев совместно используемых данных.

4. *Клиент-сервер*. Эта структура предназначена для систем, сконструированных в виде группы взаимодействующих клиентов и серверов. В качестве компонентов в данном случае выступают клиенты и серверы, а соединителями являются протоколы и сообщения, которыми они обмениваются в процессе обеспечения работоспособности системы. Такая структура требуется для разделения задач, физического распределения и выравнивания нагрузок.

Структуры распределения

1. *Размещение*. Структура размещение отражает распределение программной системы между элементами аппаратной обработки (компьютерами) и передачи данных. Отношения устанавливаются по распределению и демонстрируют физические устройства, на которых размещаются программные элементы. Возможны также отношения миграции в случае динамического распределения, например, в системах виртуальных машин. Настоящее представление позволяет инженерам анализировать производительность, целостность данных, готовность и безопасность. Все эти характеристики чрезвычайно важны в условиях распределенных и параллельных систем.
2. *Реализация*. Данная структура демонстрирует отображение программных элементов (обычно модулей) на файловую структуру (структуры) в условиях разработки системы, интеграции и управления конфигурациями. Это крайне важно в контексте разработки и процессов конструирования.
3. *Распределение функций*. Данная структура обеспечивает распределение обязанностей по реализации и интеграции модулей между соответствующими группами разработчиков. Наличие в составе архитектуры структуры распределения функций делает очевидным, что при принятии соответствующих решений учитывались как архитектурные, так и организационные факторы. Архитектору должно быть известно, какие именно навыки требуются от разных групп разработчиков.

На основе изложенного необходимо определить состав, назначение и интерфейсы архитектурных компонент программной системы, например в следующем виде:

<Наименование модуля>
Назначение модуля: …
Варианты использования через состав методов модуля на псевдокоде: <Наименование метода_1> <Наименование метода_2> <Наименование метода_3> …
Структуры и типы данных модуля: <Имя_поля_1> <Имя_поля_2> <Имя_поля_3> …
Декомпозиция: <является_подмодулем> или <связан_с> …
Класс или обобщение: <является подклассом> …

<Наименование компонент> (<Наименование соединителя>)
Назначение компонента (соединителя): …
Совместно используемые данные: <Структура или тип данных_1> <Структура или тип данных_2> <Структура или тип данных_3> …
Процесс: <Имя_процесса_1> <Имя_ процесса _2> <Имя_ процесса _3> …
Параллелизм: < Имя_потока_1> < Имя_потока_2> < Имя_потока_3> …
Клиент-сервер: <Компонент (серверный или клиентский)> <Соединитель (протоколы или сообщения обмена)> …

<Наименование структуры распределения>
Назначение структуры распределения: …
Размещение: <Наименование компонента (модуля)_1> –> <Элемент размещения_1> <Наименование компонента (модуля)_1> –> <Элемент размещения_2> <Наименование компонента (модуля)_2> –> <Элемент размещения_3> …
Реализация: <Наименование компонента (модуля)_1> –> <Путь размещения_1> <Наименование компонента (модуля)_1> –> <Путь размещения_1> <Наименование компонента (модуля)_1> –> <Путь размещения_1> …
Распределение функций: <Компетенция (специалист, навыки)_1> –> <Наименование компонента_1> <Компетенция (специалист, навыки)_2> –> <Задача интеграции_1> …

Раздел №2. Определение варианта архитектуры и ее разработка

Архитектура, основанная на уровнях абстракций

Ввиду важности концепции абстракций и широкого использования этой концепции в разработке архитектур различных сложных систем остановимся на этой концепции достаточно подробно. Наиболее мощный инструмент, дающий разработчику возможность справиться со сложностью системы – это понятие абстракции. Абстракция представляет собой описание системы или ее части, в котором не указываются абсолютно все детали. Она обеспечивает разработчику общий “макроскопический” обзор системы, в силу чего дает представление о глобальных связях и свойствах основных элементов системы, которое трудно достичь, когда учитываются все подробности. Не существует такого понятия, как вполне определенная, единственная абстракция конкретной системы. Для каждой системы имеется множество возможных абстракций. Каждая абстракция дает разработчику различную точку зрения на то, как выглядит система и что она делает. Абстракции системы могут быть тесно связаны. Так, например, одна абстракция a(2) может быть уточнением абстракции a(1). Если абстракция a(2) уточняет абстракцию a(1), то абстракция a(2) описывает все, что описывает абстракция a(1), но уровень детализации в абстракции a(2) всегда не меньше, чем в абстракции a(1). Другими словами абстракция a(2) является равномерно более детализированным описанием, чем a(1) [4].

Пусть a(1), a(2),..., a(n) – такой ряд абстракций, что для каждого i абстракция a(i + 1) – уточнение предыдущей абстракции a(i). Каждая абстракция a(i) представляет собой полное, хотя и не обязательно подробное описание всей системы. Когда абстракции упорядочены таким образом, имеет смысл говорить об уровне абстракции, т.е. о степени детализации описания. Концепция уровней абстракции была предложена Дейкстрой в 1968году.

Программа разбивается на различные иерархически упорядоченные части L(1), L(2),...,L(n), называемые слоями (уровнями), удовлетворяющие определенным проектировочным критериям. Каждый уровень является группой тесно связанных модулей. Идея уровней призвана минимизировать сложность системы за счет такого их определения, которое обеспечивает высокую степень независимости уровней друг от друга. Это достигается благодаря тому, что свойства определенных объектов такой системы (ресурсы, представления данных и т. п.) упрятываются внутрь каждого уровня, что позволяет рассматривать каждый уровень как абстракцию этих объектов.

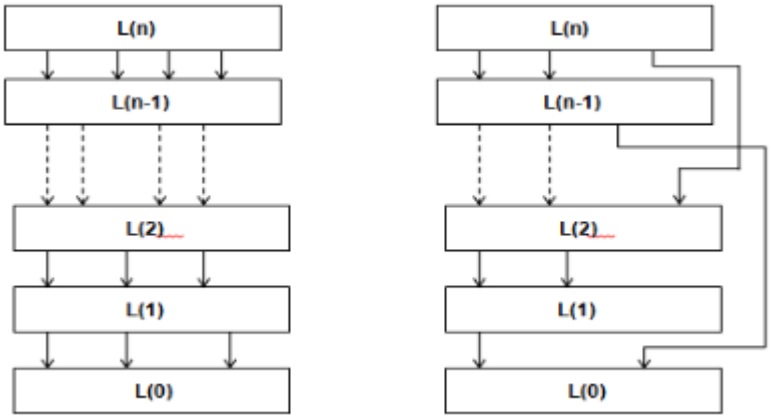

Рис. 13. Варианты классической и произвольной структуры

Архитектура, основанная на портах

В отличие от метода уровней абстракции, где внимание в первую очередь уделяется правильному распределению функций между уровнями иерархии в системе, здесь основное внимание обращено на методы связи между подсистемами. Механизмы, обеспечивающие взаимодействие между частями системы, можно разбить на три группы. Механизмы первой группы, такие как предложение GOTO, прерывание, системный вызов, передают управление от одной части программы другой, не передавая явно никаких данных. Механизмы типа вызова процедуры или подпрограммы относятся ко второй группе; они передают от одной части системы к другой и управление и данные. К третьей группе относятся механизмы портов – это методы передачи данных без передачи управления. При использовании механизма портов программная система делится на несколько подсистем, каждая из которых может иметь один или несколько входных портов (одну или несколько входных очередей). Порт это просто текущий список входных сообщений (список параметров) для подсистемы. Каждая подсистема рассматривается как асинхронный процесс, т.е. все подсистемы работают параллельно. Если одна из них хочет передать некоторые данные другой, она посылает их во входной порт этой другой подсистемы. Если подсистема готова обрабатывать какие-то данные, она читает их из одного из своих входных портов. Для организации такой связи используются операции ПОСЛАТЬ и ПОЛУЧИТЬ [4].

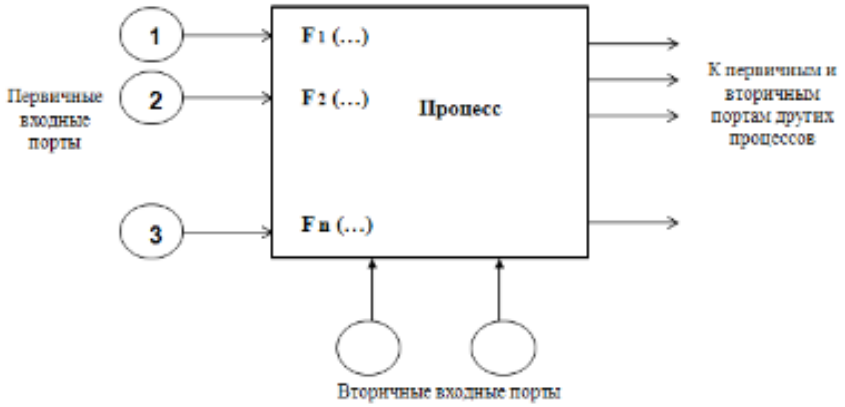

Рис. 14. Архитектура системы на основе портов прямой конфигурации

Процесс может послать данные другим процессам в результате выполнения команд ПОСЛАТЬ_УПРДАННЫМИ (X, У), по которой из аргументов X, У формируется сообщение, пересылаемое в порт по имени УПРДАННЫМИ. У процесса может быть и другой набор портов – вторичные порты. Они используются для связи с подчиненными процессами. Например, подчиненный процесс может поставлять данные из базы данных [4].

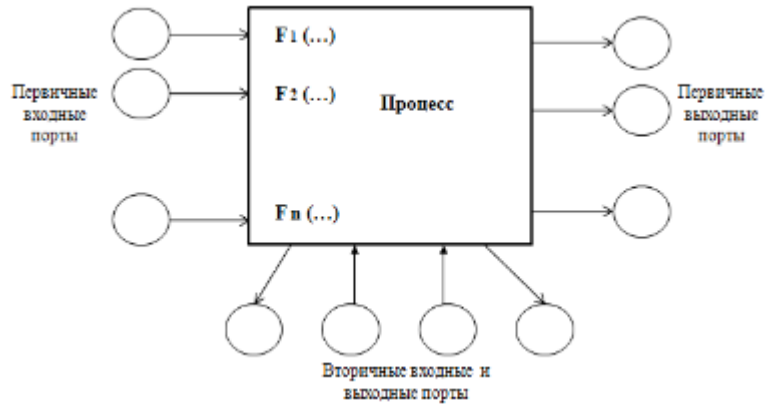

Рис. 15. Архитектура системы на основе портов непрямой конфигурации

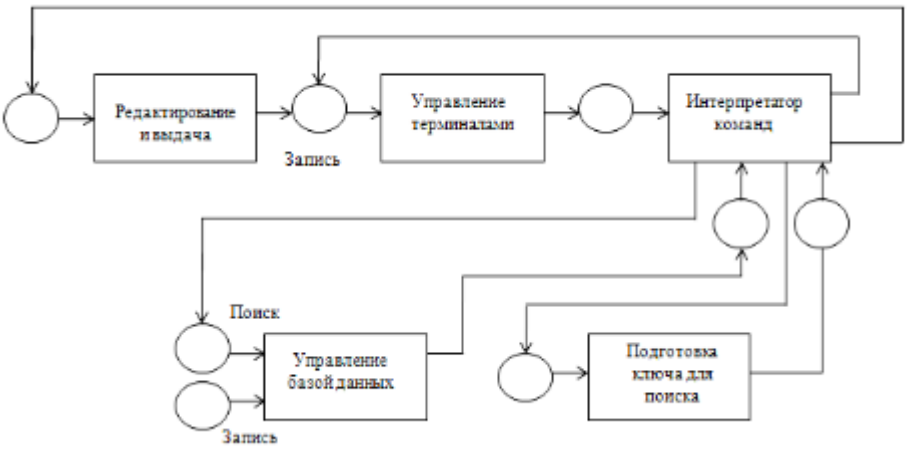

Рис. 16. Информационно-поисковая система прямой конфигурации

Архитектура, основанная на потоках данных

По сути, эти архитектуры являются современным вариантом рассмотренных архитектур, управляемых портами сообщений. В общем случае архитектуры, основанные на потоках данных, представляются как совокупность процессов, каждый из которых принимает данные из различных источников, а также возвращает данные в другие источники или хранилища данных. Процессы проектируются независимо друг от друга. Такие архитектуры давно используются и изображаются с помощью диаграмм потоков данных (DFD-диаграммы). Метамодель диаграммы потоков данных приведена на рис. 17.

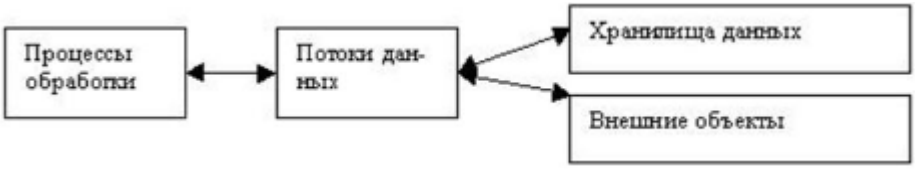

Рис. 17. Метамодель диаграммы потоков данных

Как разновидность таких архитектур рассматриваются архитектуры типа каналы и фильтры, в которых процессы называются фильтрами, а потоки направляются через каналы.

Архитектура независимых компонентов

Программные системы такой архитектуры состоят из независимо работающих компонентов, время от времени общающихся друг с другом. Компонент, называемый клиентом, выдает запрос для выполнения какого-либо действия к другому компоненту, называемому сервером. Такое отношение называется клиент-серверным. Интерфейс сервера должен быть собран в одном месте и четко определен. Преобладание таких отношений характерно для клиент-серверной архитектуры. Классическая двухуровневая архитектура клиент-сервер постоянно усложнялась добавлением новых уровней. Отдельный уровень может управлять процедурами (сервер приложений), другой – данными (сервер баз данных в трехуровневой архитектуре).

Значительную роль в разработке систем на основе архитектуры независимых компонентов сыграл международный консорциум OMG (Object Management Group). Дело в том, что в мире существует громадное количество готовых к использованию информационно-вычислительных ресурсов. Они создавались в разное время, для их разработки использовались разные подходы. Почти всегда при разработке новой информационной системы можно найти подходящие по своим функциям уже работающие готовые компоненты. Проблема состоит в том, что при их создании не учитывались требования интероперабельности (способность к взаимодействию). Эти компоненты не понимают один другого, они не могут работать совместно.

Желательно иметь механизм или набор механизмов, которые позволят сделать такие независимо разработанные информационно-вычислительные ресурсы интероперабельными [4, 7].

Одно из наиболее признанных решений проблемы унаследованных систем основывается также на объектно-ориентированном подходе. Основной задачей OMG является разработка архитектуры и методов реализации программного обеспечения, которое в объектно-ориентированном стиле позволило бы выполнить интеграцию существующих и заново разрабатываемых (не обязательно объектно-ориентированных) информационно-вычислительных ресурсов. OMG регулярно выпускает спецификационные документы, которые становятся фактическими промышленными стандартами. Основными составляющими подхода OMG являются базовая объектная модель (Core Object Model), эталонная модель архитектуры OMA (Object Management Architecture) и более приближенная к реализации общая архитектура брокера объектных заявок CORBA (Common Object Request Broker Architecture).

На рисунке 18 приведена эталонная модель метаобъектного средства OMG. На нижнем уровне находятся отдельные объекты. Объектные типы определяют общие свойства схожих объектов. Объектные типы в свою очередь являются экземплярами метаобъектной модели. С помощью этой метамодели описываются архитектуры OMG/CORBA, Microsoft/COM и Java/RMI.

Рис. 18. Эталонная модель метаобъектного средства OMG

Идея CORBA довольно проста. Она заключается в следующем. Во-первых, в каждый объект, который должен быть включен в интегрированную объектную систему, добавляется специальный программный код, обеспечивающий принципиальную возможность взаимодействия объектов. Этот код генерируется автоматически за счет использования определенного OMG языка определения объектных интерфейсов IDL (Interface Definition Language). В исходный текст программы включаются спецификации интерфейса на языке IDL. Затем этот текст должен быть обработан специальным прекомпилятором, который и генерирует дополнительный программный код. Заметим, что на сегодняшний день в документах OMG определены правила встраивания конструкций IDL в далеко не все языки программирования, но, по крайней мере, определены правила встраивания для таких популярных языков как C и C++.

Во-вторых, для реального взаимодействия должным образом настроенных объектов предполагается наличие специального программного обеспечения, называемого в документах OMG брокером объектных заявок ORB (Object Request Broker). Брокер объектных заявок должен существовать и на стороне вызывающего объекта, и на стороне вызываемого объекта. Основная задача ORB состоит в том, чтобы доставить заявку на вызов метода вызываемого объекта и возвратить результаты выполнения метода вызываемому объекту.

В настоящее время CORBA является индустриальным стандартом, описывающим высокоуровневые средства поддержки взаимодействия объектов в распределенных гетерогенных средах. Этот стандарт специфицирует инфраструктуру взаимодействия компонентов на представительском уровне и уровне приложений модели OSI, позволяя рассматривать все приложения в распределенной системе как объекты. Практика показывает, что большинство объектов одновременно исполняют роль и клиентов и серверов, позволяя тем самым с помощью CORBA, строить гораздо более гибкие системы, чем системы клиент-сервер, основанные на двухуровневой и трехуровневой архитектуре.

Сервис-ориентированная архитектура

В последнее время быстро набирает вес и де-факто становится стандартом для создания распределенных приложений технология Web-сервисов. В основе SOA лежат принципы многократного использования функциональных элементов информационных технологий, ликвидации дублирования функциональности в программной системе, унификации типовых операционных процессов, обеспечения перевода операционной модели компании на централизованные процессы и функциональную организацию на основе промышленной платформы интеграции.

Компоненты программы могут быть распределены по разным узлам сети, и предлагаются как независимые, слабо связанные, заменяемые сервисы-приложения. Программные комплексы, разработанные в соответствии с SOA, часто реализуются как набор веб-сервисов, интегрированных при помощи известных стандартных протоколов (SOAP, WSDL, и т. п.).

В принципе архитектуру этого типа можно отнести к архитектуре независимых компонент. Основное достоинство такого рода сервисов – это простота создания. Упрощается и взаимодействие между компонентами, поскольку основным транспортным протоколом является HTTP. Приложение может без проблем работать не только в Intranet-, но и в Internet-сетях. Кроме того, эта технология не привязана к какой-либо одной платформе, что открывает возможность создания распределенных приложений в гетерогенных средах. Технология интегрирована с технологией DOT.NET.

Ключевой аспект, который отличает Web-сервисы от протоколов передачи двоичных данных в моделях программирования таких, как CORBA

и DCOM: Web-сервисы – это технология работы с сообщениями, в которой передача сообщений основана на XML (SOAP), и сами Web-сервисы описываются на XML (WSDL). Логическим продолжением технологии Web-сервисов стала архитектура, ориентированная на сервисы (SOA) [4].

Ее появление является следствием новых задач и потребностей, возникающих при создании и эксплуатации современных автоматизированных/информационных систем:

- оперативное реагирование на изменение требований и быстрая адаптация к новым задачам;
- оптимизация управления бизнес-процессами;
- эффективное обеспечение внешних взаимодействий.

К основным характеристикам SOA, отличающим ее от других архитектур автоматизированных/информационных систем, следует отнести:

- распределенность;
- слабосвязанные интерфейсы (отсутствие жестких связей между элементами системы), что упрощает конфигурирование системы и координирование работы ее элементов;
- базирование архитектуры на международных открытых стандартах;
- возможность динамического поиска и подключения нужных функциональных модулей;
- построение систем с расчетом на процессы с использованием сервисов, ориентированных на решение определенных бизнес-задач.

Метамодель SOA (рис. 2.19) включает следующие модели:

- модель политик (PM) определяет политики, связанные с архитектурой, в основном она представляет собой ограничения, накладываемые на поведение агентов и сервисов, в том числе ограничения, связанные с требованиями безопасности и качества обслуживания;
- модель, ориентированная на сервисы (SOM), сосредоточена на действиях, выполняемых сервисами;
- модель, ориентированная на ресурсы (ROM), сосредоточена на системных ресурсах;
- модель, ориентированная на сообщения (MOM), сосредоточена на сообщениях, их структуре, способах транспортировки и других компонентах, никак не связанных с причинами обмена сообщениями и их семантикой.

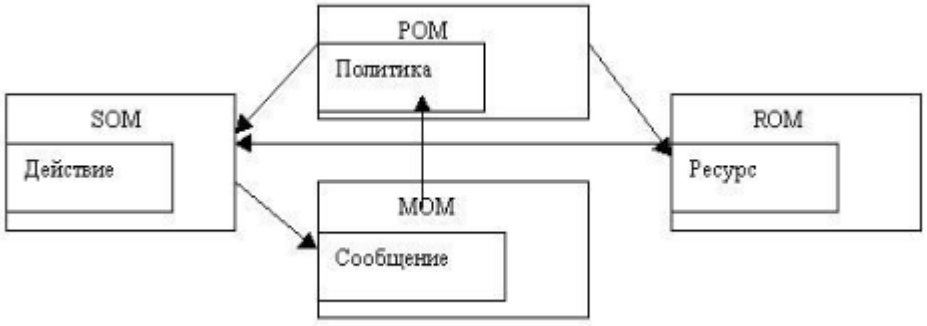

Рис. 19. Метамодель SOA

Важную роль в этой архитектуре отводится управлению Web-сервисами, к основным функциям которого относятся развертывание, конфигурирование, обеспечение безопасности, а также функции мониторинга и диагностики. Уже реализуется скоординированное движение к обеспечению управления Web-сервисами как ресурсом. Стандарт, к которому стремится большинство поставщиков, – это Web Services for Distributed Management (WSDM). В стандарт входит набор Web-сервисов и операций, которые должна будет обеспечивать каждая реализация Web-сервиса, – операции для конфигурирования, мониторинга и других задач управления.

При необходимости внести изменения в интерфейс сервиса и провайдер, и пользователи Web-сервиса должны изменить свои среды одновременно, что может оказаться затруднительным. Однако имеется возможность (и она реализована) ввести "посредника" между потребителем сервиса и его провайдером, что существенно повышает адаптируемость и масштабируемость систем.

В сервисной архитектуре просматриваются черты традиционных архитектур. С помощью сценарных языков типа BPEL можно описывать новые сервисы или описывать бизнес-процессы с участием сервисов (виртуальная машина). Системы, реализованные в других стандартах (например, CORBA) легко интегрируются в SOA (соответствующий инструментарий имеется).

SOA представляет собой не только основу для новых информационных технологий, но также и новую системную философию. Постепенный переход от мэйнфреймов к конфигурациям типа "клиент-сервер" и далее к SOA означает глубокие изменения в функциональности систем, и, следовательно, создает новые возможности в областях применения информационных технологий. SOA имеет все шансы заменить со временем монолитные корпоративные системы, сложные как во внедрении, так и в модернизации.

Раздел №3. Заключение и выводы

В данном разделе необходимо привести пронумерованный перечень выводов, которые были сделаны в результате разработки архитектурных структур и выбора варианта архитектуры.

Коллективное выполнение лабораторной работы

За каждым из участников команды из трех человек должна быть зафиксирована роль, в соответствии с которой должны быть решены задачи, определенные в графе «Зона ответственности».

Роль	«Специалист по научно-исследовательским и опытно-конструкторским разработкам»
Зона ответственности	Разработка документов: – документ о концепции и границах проекта; – спецификация требований к программному обеспечению.
Приоритет при защите	первый

Роль	«Системный аналитик»
Зона ответственности	Разработка документов: – модель предметной области.
Приоритет при защите	второй

Роль	«Архитектор программного обеспечения»
Зона ответственности	Разработка документов: – архитектура программного обеспечения.
Приоритет при защите	третий

Подготовка к защите лабораторной работы

Каждый участник команды из трех человек должен по результатам проведенной работы подготовить устный доклад на 3-5 минут, в рамках которого он должен пояснить, что было сделано, какие возникли сложности, проблемы в реализации этапов и какие были сделаны выводы в процессе их реализации.

Специалист по научно-исследовательским и опытно-конструкторским разработкам должен быть готов ответить на следующие вопросы.

1. Какие проблемы в предметной области были идентифицированы в процессе разработки документа?
2. Какими источниками информации пользовались для описания предметной области?

3. В чем состоит обоснование состава вариантов использования программной системы?

Системный аналитик должен быть готов ответить на следующие вопросы.

1. Какие ограничения предметной области учитываются?
2. Какое назначение языков моделирования при проектировании моделей предметной области?
3. Какие элементы документа о концепции и границах проекта использовались для моделирования процессов в предметной области?

Архитектор программного обеспечения должен быть готов ответить на следующие вопросы.

1. В чем заключается обоснование выбранного варианта архитектуры?
2. Какие элементы спецификации требований использовались для разработки архитектуры?
3. По какому принципу формировался состав модулей архитектуры программной системы?

Список литературы

1. *Вигерс К., Битти Д.* Разработка требований к программному обеспечению/Пер. с англ. – М.: Издательство "БХВ-Петербург", 2014. – 736 с.: ил.
2. Графический язык моделирования бизнес-процессов BPMN Версия 2.0. Пер. с англ. комп. EleWise [Электронный ресурс]. URL: elma-bpm.ru (дата обращения 10.09.2021). 298 с.
3. *Купер А., Рейман Р., Кронин Д.* Алан Купер об интерфейсе. Основы проектирования и взаимодействия. – Пер. с англ. – СПб.: Символ-Плюс, 2009. – 688 с., ил.
4. *Назаров С. В.* Архитектура и проектирование программных систем [Электронный ресурс] : Монография / С.В. Назаров. - М.: НИЦ ИНФРАМ, 2014. - 351 с. — Режим доступа: http://znanium.com/bookread2.php?book=542562 — Загл. с экрана.
5. *Петров В.* Алгоритм решения изобретательских задач. Учебное пособие. Тель-Авив, 1999. ISBN 965-7127-00-9.
6. *Фаулер, М.* UML. Основы : Краткое руководство по стандартному языку объектного моделирования [Текст] / М. Фаулер ; авт. предисл. К. Кобрин [и др.]. - 3-е изд. - СПб. : Символ, 2014. - 192 с. : рис.
7. *Фаулер, М.* Шаблоны корпоративных приложений [Текст] = Patterns of enterprise application architecture : пер. с англ / М. Фаулер ; соавт. Д. Райс [и др.]. - М. : Вильямс, 2014. - 544 с. : рис.

Варианты заданий для выполнения лабораторных работ

Номер варианта задания	Задание на лабораторную работу
1	Разработка программного обеспечения для автоматизированной/информационной системы железной дороги
2	Разработка программного обеспечения для автоматизированной/информационной системы авиакомпании
3	Разработка программного обеспечения для автоматизированной/информационной системы аэропорта
4	Разработка программного обеспечения для автоматизированной/информационной системы морского порта
5	Разработка программного обеспечения для автоматизированной/информационной системы автобусного вокзала
6	Разработка программного обеспечения для автоматизированной/информационной системы школы
7	Разработка программного обеспечения для автоматизированной/информационной системы библиотеки
8	Разработка программного обеспечения для автоматизированной/информационной системы университета
9	Разработка программного обеспечения для автоматизированной/информационной системы службы занятости
10	Разработка программного обеспечения для автоматизированной/информационной системы службы социальной защиты
11	Разработка программного обеспечения для автоматизированной/информационной системы поликлиники
12	Разработка программного обеспечения для автоматизированной/информационной системы обязательного медицинского страхования
13	Разработка программного обеспечения для автоматизированной/информационной системы пенсионного фонда
14	Разработка программного обеспечения для автоматизированной/информационной системы выставочного комплекса
15	Разработка программного обеспечения для автоматизированной/информационной системы для организации НИОКР

16	Разработка программного обеспечения для автоматизированной/информационной системы издательства
17	Разработка программного обеспечения для автоматизированной/информационной системы редакции газеты
18	Разработка программного обеспечения для автоматизированной/информационной системы типографии
19	Разработка программного обеспечения для автоматизированной/информационной системы гостиницы
20	Разработка программного обеспечения для автоматизированной/информационной системы киноцентра
21	Разработка программного обеспечения для автоматизированной/информационной системы фирмы по прокату автомобилей
22	Разработка программного обеспечения для автоматизированной/информационной системы букмекерской фирмы
23	Разработка программного обеспечения для автоматизированной/информационной системы фондовой биржи
24	Разработка программного обеспечения для автоматизированной/информационной системы банка
25	Разработка программного обеспечения для автоматизированной/информационной системы лизинговой компании
26	Разработка программного обеспечения для автоматизированной/информационной системы туристического агентства
27	Разработка программного обеспечения для автоматизированной/информационной системы фильмотеки
28	Разработка программного обеспечения для автоматизированной/информационной системы агентства недвижимости
29	Разработка программного обеспечения для автоматизированной/информационной системы страховой организации
30	Разработка программного обеспечения для автоматизированной/информационной системы автошколы
31	Разработка программного обеспечения для автоматизированной/информационной системы оператора связи
32	Разработка программного обеспечения для автоматизированной/информационной системы автосервиса
33	Разработка программного обеспечения для автоматизированной/информационной системы для оказания госуслуг

34	Разработка программного обеспечения для автоматизированной/информационной системы фирмы по сборке и продаже компьютеров и комплектующих
35	Разработка программного обеспечения для автоматизированной/информационной системы транспортной фирмы
36	Разработка программного обеспечения для автоматизированной/информационной системы супермаркета
37	Разработка программного обеспечения для автоматизированной/информационной системы книжного магазина
38	Разработка программного обеспечения для автоматизированной/информационной системы ломбарда
39	Разработка программного обеспечения для автоматизированной/информационной системы ГИБДД
40	Разработка программного обеспечения для автоматизированной/информационной системы спортивного клуба
41	Разработка программного обеспечения для автоматизированной/информационной системы интернет-провайдера
42	Разработка программного обеспечения для автоматизированной/информационной системы интернет-магазина
43	Разработка программного обеспечения для автоматизированной/информационной системы интернет-аукциона
44	Разработка программного обеспечения для автоматизированной/информационной системы почтовой службы
45	Разработка программного обеспечения для автоматизированной/информационной системы предприятия ЖКХ
46	Разработка программного обеспечения для автоматизированной/информационной системы рекламного агентства
47	Разработка программного обеспечения для автоматизированной/информационной системы курьерской фирмы
48	Разработка программного обеспечения для автоматизированной/информационной системы ресторанного комплекса
49	Разработка программного обеспечения для автоматизированной/информационной системы службы такси
50	Разработка программного обеспечения для автоматизированной/информационной системы службы технической поддержки
51	Разработка программного обеспечения для автоматизированной/информационной системы <свой вариант> (необходимо сформулировать тему)