Практическое занятие. Построение физической модели АСОИУ.

Цель занятия: научиться формировать диаграмму компонентов и диаграмму развертывания для формирования физической модели процесса.

Теоретические сведения к практическому занятию

Диаграммы компонентов используются при моделировании физических аспектов объектно-ориентированных систем, которые используются для визуализации, определения и документирования компонентных систем, а также для создания исполняемых систем посредством прямого и обратного проектирования. Диаграммы компонентов - это, по сути, диаграммы классов, которые фокусируются на компонентах системы, которые часто используются для моделирования статического представления системы.

Диаграммы компонентов используются для визуализации организации компонентов системы и отношений зависимости между ними. Они обеспечивают общее представление о компонентах системы.

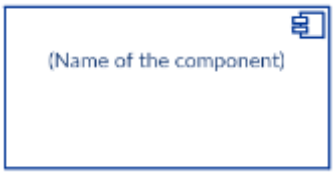
Компоненты могут быть программным компонентом, таким как база данных или пользовательский интерфейс; или аппаратный компонент, такой как схема, микрочип или устройство; или бизнес-подразделение, такое как поставщик, платежная ведомость или отгрузка.

Основными графическими элементами диаграммы являются компоненты, интерфейсы и зависимости между ними.

Есть три способа использования символа компонента.

Таблица 1

Способы использования символа компонента

Компонента	Изображение
Прямоугольник со стереотипом компонента <<component>>	
Прямоугольник со значком компонента в правом верхнем углу и названием компонента	
Прямоугольник со значком компонента и стереотипом компонента	

Интерфейсы на диаграммах компонентов показывают, как компоненты связаны друг с другом и взаимодействуют друг с другом. Коннектор сборки позволяет связать требуемый интерфейс (представленный полукругом и сплошной линией) с предоставленным интерфейсом (представленный кругом и сплошной линией) другого компонента. Это показывает, что один

компонент предоставляет услугу, которая требуется другому.

Рис. 1. Один компонент предоставляет услугу, которая требуется другому

Порт (представленный маленьким квадратом в конце требуемого интерфейса или предоставленного интерфейса) используется, когда компонент делегирует интерфейсы внутреннему классу.

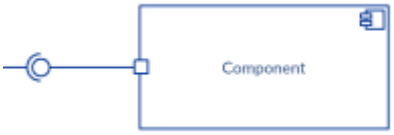


Рис. 2. Порт

Хотя вы можете показать более подробную информацию о взаимосвязи между двумя компонентами, используя нотацию «шарик-и-сокет» (предоставленный интерфейс и требуемый интерфейс), вы также можете использовать стрелку зависимости, чтобы показать взаимосвязь между двумя компонентами.

Рис. 3. Зависимости

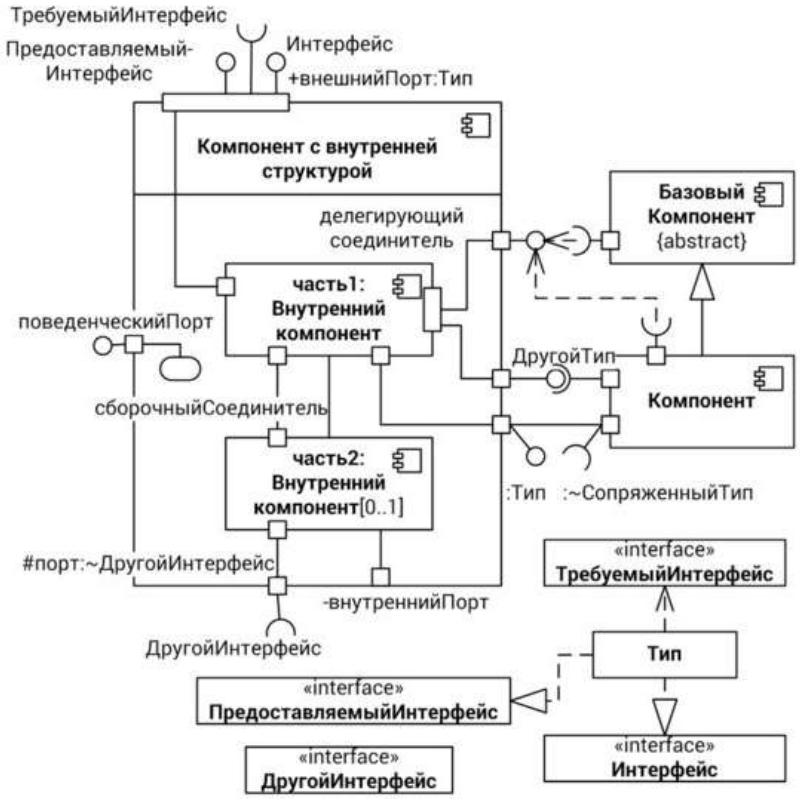


Рис. 4. Нотация диаграмм компонентов

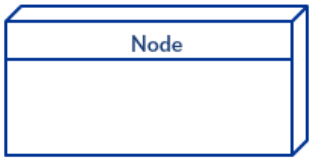
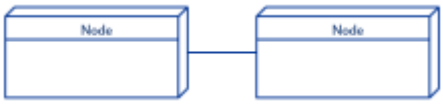
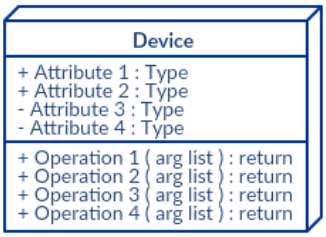
Диаграмма развертывания - это тип диаграммы UML, который показывает архитектуру выполнения системы, включая узлы, такие как аппаратные или программные среды выполнения, и связующее их промежуточное программное обеспечение.

Диаграммы развертывания обычно используются для визуализации физического оборудования и программного обеспечения системы. Используя его, можно понять, как система будет физически развернута на оборудовании.

Диаграммы развертывания помогают смоделировать топологию оборудования системы по сравнению с другими типами диаграмм UML, которые в основном описывают логические компоненты системы.

Таблица 2

Элементы диаграммы развертывания

Элемент	Изображение	Описание
Узлы		Узел, представленный в виде куба, представляет собой физический объект, который выполняет один или несколько компонентов, подсистем или исполняемых файлов. Узел может быть аппаратным или программным элементом.
Артефакты		Артефакты - это конкретные элементы, возникшие в процессе разработки. Примеры артефактов: библиотеки, архивы, файлы конфигурации, исполняемые файлы и т. д.
Коммуникационная ассоциация		Это представлено сплошной линией между двумя узлами. Он показывает путь связи между узлами.
Устройства		Устройство - это узел, который используется для представления физического вычислительного ресурса в системе. Примером устройства является сервер приложений.

Спецификации развертывания	Deployment Specification + Attribute 1 : Type + Attribute 2 : Type - Attribute 3 : Type - Attribute 4 : Type	Спецификации развертывания - это файл конфигурации, например текстовый файл или XML-документ. Он описывает, как артефакт развертывается на узле.

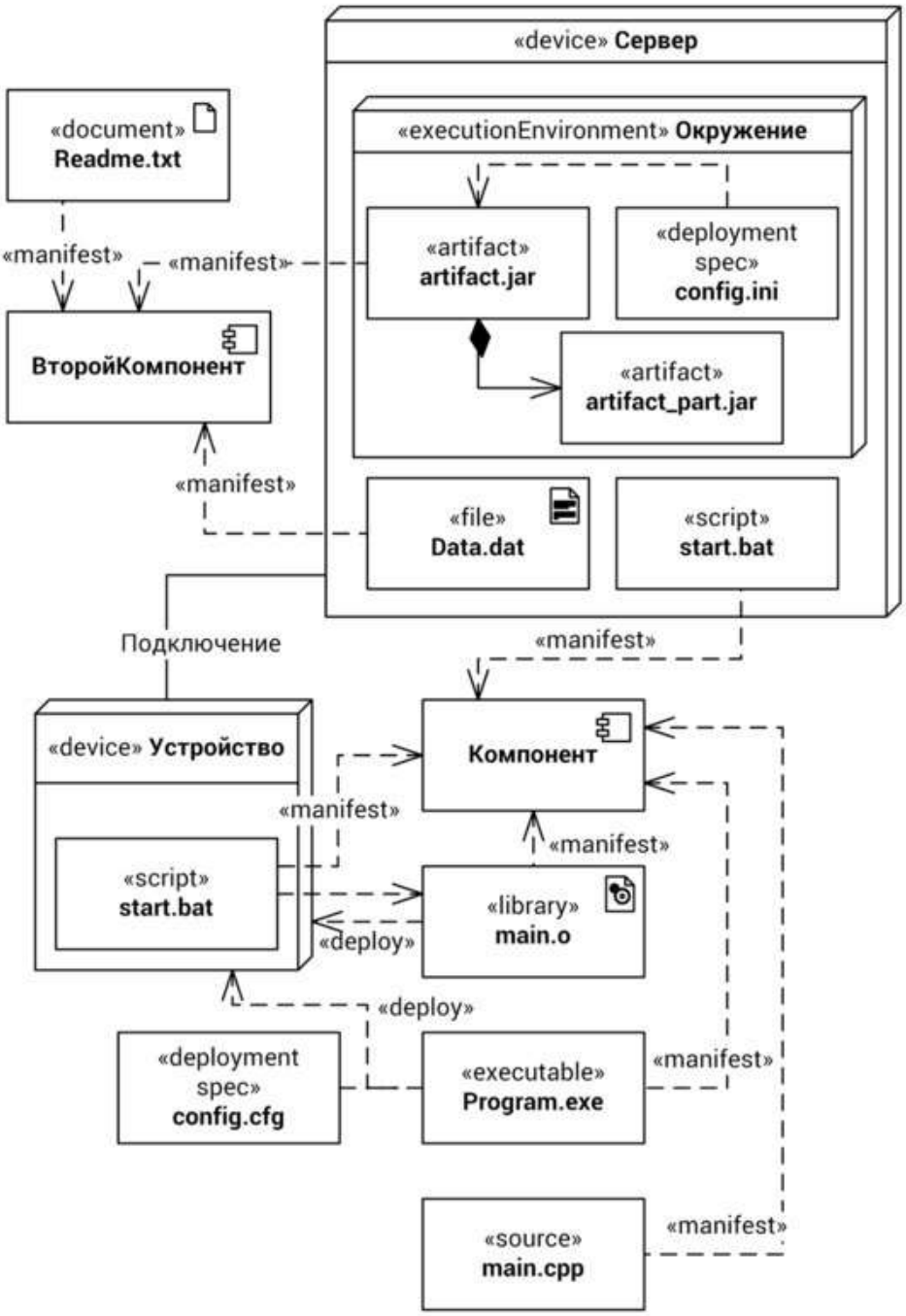


Рис. 8. Нотация диаграмм развертывания

На рисунке 9 показан пример диаграммы развертывания программно-технологического комплекса ИСКРА на отдельной железной дороге.

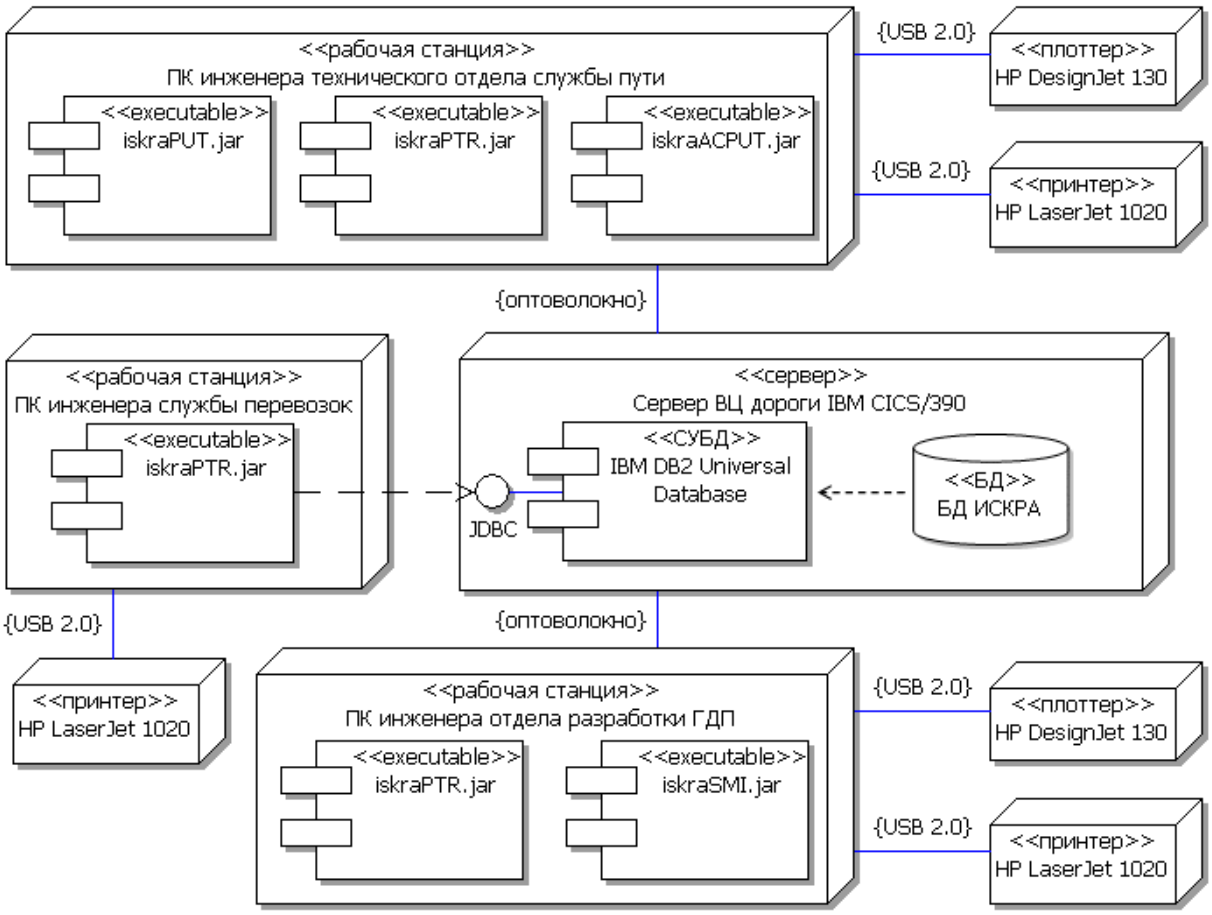


Рис. 9. Пример диаграммы развертывания с компонентами и интерфейсами

Задачи для самостоятельной работы

10.1. Подсистема подготовки данных модуля морфологии *MorphologyDPS* состоит из базы данных *Database*, клиента для модификации данных *DataClient*, компонента экспорта *Export* и компилятора данных *Compiler*.

а. База данных предоставляет интерфейс изменения данных *IMorphologyData* и интерфейс экспорта данных *IDataExport*. Клиент требует для работы интерфейс изменения данных, в то время как компонент экспорта требует интерфейс экспорта данных. Компилятор не требует внешних интерфейсов, но неявно зависит от базы данных. Укажите в модели, как компоненты связаны между собой в подсистеме.

б. Разместите базу данных на сервере *MorphoDB*, а остальные компоненты на компьютере лингвиста *LinguistWorkPlace*.

в. Уточните внутреннюю структуру компилятора следующим образом. Компилятор использует интерфейс *IMorphology* компонента *MorphoModel*. Сам компилятор состоит из парсера *Parser*, обработчика сообщений об ошибках *Handler* и сборщика модели *Builder*. Компоненты, реализующие парсер и сборку моделей, сообщают об ошибках через интерфейс *IErrorHandler* компонента *Handler* в составе компилятора. Сборщик модели компилятора требует внешний интерфейс *IMorphology*.

10.2. Приложение класса *Application* содержит подключаемые модули. Подключаемый модуль класса *Bean* является либо процессным модулем *ProcessorBean*, либо алгоритмическим модулем *ComputeBean*. Процессный модуль связан *ComputeLink* с подключаемыми модулями для выполнения расчетов.

а. Используя представление внутренней структуры, укажите, что специализация *MainApp* приложения *Application* включает один процессный модуль и два связанных с ним алгоритмических модуля.

б. Доработайте модель, укажите, что приложение *MainApp* включает два связанных процессных модуля, один из которых является основным *main*.

в. Покажите, что основной процессный модуль приложения *MainApp* реализует интерфейс управления вычислениями *Computation*, предоставляемый приложением через порт веб-сервисов *ComputationEndpoint*.

г. Укажите, что приложение *MainApp* предоставляет и реализует интерфейс конфигурации *Configuration* через порт *ConfigurationEndpoint*.

д. Используя соединители сборки, покажите, что основной процессный модуль приложения *MainApp* может обращаться через интерфейс *Computation* к приложению *SecondApp*.

10.3. Для системы обработки текстов выбран архитектурный стиль Pipes-and-Filters. Одна из функций системы – векторизация текста – включает чтение текста из файла, очистку текста от служебных символов и знаков пунктуации, разделение на слова, нормализацию и вычисление частоты вхождения в текст каждого слова и запись результатов в файл. Размещение системы предполагается на серверах *Srv1* и *Srv2* с ОС Linux, соединенных вычислительной сетью.

а. Покажите на диаграмме классов структуру системы, если известно, что фильтры бывают только обработчиками *Processor*, источниками *Source* или результатами *Result*.

б. Воплотите компоненты для функции векторизации текста в исполняемых файлах и разместите их на сервере *Srv1*.

в. Добавьте в компоненты системы возможность сбора статистики через интерфейс *Status* агентами сбора статистики *Monitor*. Разместите компоненты вместе с агентами и управляющим компонентом статистики на двух серверах *Srv1* и *Srv2* так, чтобы нормализация текста проводилась отдельно на сервере *Srv2*, где также установлен управляющий компонент статистики.

10.4. Удаленный отладчик для встраиваемых программ *TargetDebugger* позволяет собирать треки выполнения программы на устройстве, передавать их для сопоставления с UML моделью программы на рабочей станции разработчика. Отладчик состоит из программы-монитора на устройстве *DeviceMonitor* и модуля подключения по сети *RemoteLib*.

а. Интерфейс *Monitor* монитора позволяет указать *setBreakpoint* точку останова по адресу в памяти *addr*, остановить *pause* и продолжить *resume* выполнение программы. Покажите

на диаграмме классов интерфейс монитора, и его реализации для разных платформ *ARM7*, *PowerPC* и *Cortex-M3*.

б. Покажите структуру модуля подключения по сети, используя бинарный протокол *BinaryTcp* и текстовый протокол *Telnet*. Воспользуйтесь паттерном *Proxy* и представьте решение на диаграмме компонентов. Решение поясните.

в. Выберите артефакты и разместите модули отладчика на рабочем месте разработчика и устройстве на *ARM7*, предполагая связь по *Telnet*. В каком архитектурном стиле реализован отладчик?