ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им. А.Н. КОСЫГИНА
(ТЕХНОЛОГИИ. ДИЗАЙН. ИСКУССТВО)

На правах рукописи

Гусев Александр Олегович

РАЗРАБОТКА КОНЦЕПЦИИ СИСТЕМЫ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ ОБУВИ С ПРИМЕНЕНИЕМ ОБЛАЧНЫХ ТЕХНОЛОГИЙ

Специальность 05.19.05

«Технология кожи, меха, обувных и кожевенно-галантерейных изделий»

ДИССЕРТАЦИЯ

на соискание ученой степени

кандидата технических наук

Научный руководитель

доктор технических наук, профессор

Костылева В.В.

Москва – 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность темы. Обувная промышленность является значимой отраслью национальной экономики, она играет важную роль в обеспечении стабильного экономического роста и направлена на улучшение качества жизни населения [1]. Для решения проблем отрасли Министерством промышленности и торговли была разработана Стратегия развития легкой промышленности в Российской Федерации на период до 2025 года [2], в которой выделена необходимость совершенствования организации производства путем внедрения новых цифровых технологий и инноваций. Цифровая трансформация в числе Национальных целей, объявленных в Указе Президента РФ от 21.07.2020 N 474 «О национальных целях развития Российской Федерации на период до 2030 года» [3].

Сегодня, для удовлетворения потребностей клиентов и сохранения конкурентоспособности, производители обуви должны решать две основные задачи: быстро реагировать на рыночные изменения и соответствовать новым потребительским тенденциям. Одним из решений этих задач является применение систем автоматизированного проектирования (САПР), позволяющих упростить процессы конструирования и моделирования, сокращая время на разработку новой продукции.

В 2020 году мировую экономику потрясло воздействие COVID-19. В первый год пандемии были введены жесткие ограничительные меры, за которыми последовала приостановка большинства производств и освоение дистанционного режима работы. Сегодня многие из введенных ограничений сняты, однако правительство время от времени возвращается к ним, в зависимости от состояния эпидемиологической ситуации. Из-за этого множество компаний не возвращается к привычному режиму работы.

Современные САПР обуви отрыты для совершенствования [4] в области организации дистанционного режима работы. Системы лицензируются по традиционной модели, которая включает в себя защиту от копирования в виде аппаратного ключа. Это создает риски для ее портативности – утрата ключа равносильна покупке новой программы. Ввиду того, что САПР обуви призваны выполнять большие объемы сложных математических вычислений и графических построений, предъявляются высокие требования к аппаратным средствам, что повышает стоимость всей системы. Используемые распространенными САПР обуви методы оцифровки неэффективны в условиях дистанционного режима работы. Покупка дигитайзеров на каждое рабочее место – экономически затратно. В свою очередь, оцифровка по изображению производится с отдельными лекалами, а не чертежом, что подразумевает, необходимость выполнения части работы вручную, которую могла бы автоматизировать САПР.

Решением проблем стали облачные технологии, которые сегодня активно развиваются [4,5]. Облачные приложения распространяются по модели Software as a Service. В рамках этой модели программное обеспечение располагается на удаленном облачном сервере поставщика, а пользователи получают к нему доступ через веб-интерфейс с любого устройства и операционной системы. Модель экономически выгодна как для поставщика, так и для пользователя. Расходы на процессы развертывания и обновления системы снижаются, так как появляется возможность на серверах поставщика применять инструменты непрерывной интеграции (Continuous Integration) и непрерывной доставки (Continuous Delivery). Также отпадает необходимость в защите от копирования, так как программа не попадает на компьютер пользователя. Поставка коммерческого программного обеспечения возможна в формате аренды, предполагающая, что, в случае временного отсутствия необходимости в программном обеспечении, пользователь может приостановить его

использование и заморозить выплаты поставщику. В отличии от традиционной схемы лицензирования пользователь несет сравнительно небольшие периодические затраты и ему не требуется инвестировать существенные средства для приобретения и поддержки программного обеспечения. Облачные технологии допускают централизацию данных, открывая новые возможности, например, такие как, система контроля версий проектов или режим совместной многопользовательской работы.

Крупные разработчики САПР общего назначения (Autodesk и Dassault Systemes) с 2012 года постепенно внедряют облачные технологии в свои системы [6]. Однако, используют облачные технологии, преимущественно для хранения данных, так как перенести всю систему в «облако» без глобальных изменений в кодовой базе невозможно. Настоящим прорывом стала система твердотельного проектирования OnShape, которая продемонстрировала возможности веб-технологий в области высокой графической нагрузки [7].

Сегодня не представлено ни одной облачной системы автоматизированного проектирования, решающей конструкторские задачи обувного производства. Существующие облачные САПР общего назначения направлены на твердотельное моделирование, не подходящее для проектирования обуви. В этой связи работа «Разработка концепции системы автоматизированного проектирования обуви с применением облачных технологий» является актуальной. Она отвечает Стратегии национальной безопасности Российской Федерации [8], Стратегии развития информационного общества в Российской Федерации [9], а также Указу Президента РФ от 1 декабря 2016 г. № 642 «О Стратегии научно-технологического развития Российской Федерации» [10].

Степень научной разработанности проблемы. Существенный вклад в решение проблем развития и совершенствования проектирования и производства обуви, повышения ее эргономических показателей качества

внесли Зыбин Ю.П., Ключникова В.М., Кочеткова Т.С., Калита А.Н., Фукин В.А., Лыба В.П., Горбачик В.Е., Бекк Н.В., Карабанов П.С. и др., в научных трудах которых разработаны методологические основы создания конструкций обуви, методы и средства их оценки, в том числе с использованием цифровых, компьютерных и информационных технологий.

Диссертационная работа отвечает формуле специальности 05.19.05 – «Технология кожи, меха, обувных и кожевенно-галантерейных изделий» – «…сложившаяся область науки и техники, включающая в себя изучение и теоретическое обоснование сущности и способов изготовления изделий легкой промышленности, обладающих необходимыми эксплуатационными и эстетическими свойствами».

В части области исследований диссертационная работа соответствует п. 12 «Разработка теоретических основ проектирования обуви, кожгалантереи и других изделий из кожи, в том числе автоматизированного» и п. 14 «Разработка теоретических основ информационных технологий в кожевенно-обувной промышленности, направленных на разработку САПР и АСУ ТП» паспорта научной специальности 05.19.05 – «Технология кожи, меха, обувных и кожевенно-галантерейных изделий».

Объект исследования – процессы проектирования конструкций обуви, системы автоматизированного проектирования изделий обувной и кожевенно-галантерейных изделий, облачные технологии.

Предмет исследования – структуры систем автоматизированного проектирования конструкций обуви и кожевенно-галантерейных изделий.

Целью диссертации является разработка концепции системы автоматизированного проектирования обуви с применением облачных технологий.

В соответствии с оставленной целью в диссертации:

- проведен анализ:
  - актуальных проблем САПР обуви;
  - возможностей облачных технологий для решения выявленных проблем САПР обуви;
  - технической составляющей облачной САПР обуви;
  - распространенных архитектурных шаблонов и их пригодность для разработки облачной САПР обуви;
- разработана архитектура облачной САПР обуви, включающая описание базовых компонентов: ядра, модуля, их структуру и способ взаимодействия;
- предложен способ реализации ядра системы, для которого разработаны:
  - подсистемы ядра: движок, конфигурация, репозитории локальных и общих временных файлов, репозиторий сервисов, реестры внутренних и внешних сервисов, подсистема запуска модулей, обработчик запросов, контейнер очереди запросов, сервер, логер, подсистемы мониторинга и обеспечения контроля доступа;
  - пошаговый процесс осуществления аутентификации входящего запроса;
  - структуры данных для описания модулей и пересылаемых сообщений;
  - формы конфигурации ядра и сервисов, описываемых текстовым форматом JSON;
  - алгоритмы синхронной и асинхронной обработки входящих сообщений, обработки собственно сообщения, формирования и отправки ответного сообщения.
- разработан метод оцифровки чертежа с использованием средств технического зрения, для которого:

  предложены:
  - этапы оцифровки – калибровка камеры, подготовка изображения, векторизация;

- способ калибровки камеры при помощи открытой библиотеки обработки изображений OpenCV;
- алгоритм подготовки изображения к оцифровке с использованием реперных точек и библиотеки OpenCV;

разработаны:

- сервис управления камерами, выполняющий калибровку камеры, и контроль за данными, полученными в результате калибровки;
- сервис оцифровки, реализующий предложенный алгоритм подготовки изображения к обработке;
- алгоритмы обработки сообщений модулями управления камерами и оцифровки;

- разработаны модели проектных данных системы и сервис их обработки;
- предложен класс систем управления базами данных для хранения проектных данных;
- сформулированы направления развития концепции облачной САПР обуви.

Исследования проводились на кафедре Художественного моделирования, конструирования и технологии изделий из кожи в рамках тематического плана НИР (2014-2018 г.), проблема 2 «Проблемно-ориентированные исследования в области перспективных технологий и дизайна», (тема 2.3 «Совершенствование методов компьютерного дизайна и проектирования, а также в рамках научно-исследовательских работ ФГБОУ ВО «РГУ им. А.Н. Косыгина» на 2019-23 г.г., проблема 1 «Матричный подход к формированию цифровой индустрии 4.0 на промышленных предприятиях текстильной и легкой промышленности», Тема 1.2 «Развитие инновационного потенциала предприятий по производству изделий из кожи на основе современных цифровых технологий проектирования и быстрого прототипирования».

Методы исследования и технические средства решения задач.

Исследования базировались на комплексном системном подходе с использованием возможностей современных информационных технологий. В ходе выполнения работы использованы теоретические положения конструирования обуви и разработки информационных систем, теория построения облачных систем.

Информационно-теоретической базой диссертации послужили труды отечественных и зарубежных ученых в исследуемой и смежных областях, энциклопедическая и справочная литература, конструкторско-технологическая документация.

В работе использовались следующие технические средства:

- система автоматизации процессов проектирования и разработки программного обеспечения Modelio;
- интегрированная среда разработки Visual Studio;
- система контейнеризации Docker;
- система оркестрации контейнеров Kubernetes;
- система тестирования сетевых программных интерфейсов Postman;
- система автоматизации развертывания облачных систем Terraform.

Научную новизну исследования составляют:

- концепция системы автоматизированного проектирования обуви с использованием облачных технологий, включающая разработку:
  - базовых компонентов облачной САПР обуви;
  - архитектуры, позволяющей ускорить разработку облачной САПР обуви и максимизировать эффективность применения облачных технологий;
  - спецификации ядра облачной САПР обуви, описывающей внутреннюю структуру и методы взаимодействия с другими компонентами;
  - спецификации модуля оцифровки облачной САПР обуви с использованием средств технического зрения, включающий описание

внутренней структуры модуля и метода взаимодействия с другими компонентами;

— модели проектных данных, в равной мере эффективных как для хранения данных в базе, так и взаимодействия со сторонними системами и отображения на экране;

- сформулированные направления развития концепции облачной САПР обуви.

Практическую значимость диссертации составляют разработанные:

- архитектура облачной САПР обуви, включающая:
  — компоненты системы и их композицию;
  — способ взаимодействия между компонентами;
- метод оцифровки в модуле облачной САПР обуви с применением средств технического зрения.

Достоверность проведенных исследований базируется на согласованности аналитических и экспериментальных результатов, использовании информационных технологий, современных методов и средств проведения исследований. Апробация основных положений диссертации производилась в научной периодической печати, конференциях, в рамках научного проекта (грант ректора ФГБОУ ВО «РГУ им. А.Н. Косыгина» 2019-2020 г.г.) «Цифровизация и обработка чертежей конструкций верха обуви с использованием средств технического зрения», а также на АО «Егорьевск-обувь» (Приложение А, Б).

Основные положения, выносимые на защиту:

- архитектура облачной САПР обуви, определяющая композицию и способ взаимодействия компонентов системы;
- спецификация ядра облачной САПР обуви;
- спецификация сервиса облачной САПР обуви;
- метод оцифровки чертежа с использованием средств технического зрения;

- расширяемая иерархичная модель данных, описывающая конструкции верха обуви.

Личный вклад автора. Автором сформулированы цель и основные задачи исследования, проанализированы распространенные САПР обуви, выявлены их проблемы, требования к распределенным и облачным системам; рассмотрены перспективы применения облачных технологий для устранения существующих проблем САПР обуви; определен архитектурный стиль системы, предложены инструменты реализации концепции САПР обуви на основе облачных технологий.

Апробация и реализация результатов работы.

Основные научные результаты проведенных исследований докладывались и получили положительную оценку на заседаниях кафедры художественного моделирования, конструирования и технологии изделий из кожи Российского государственного университета имени А.Н. Косыгина (Технологии. Дизайн. Искусство), международной научно-практической конференции «Фундаментальные и прикладные проблемы создания материалов и аспекты технологий текстильной и легкой промышленности», 14–15 ноября Казань 2019, научно-практической международной конференции «Эргодизайн как инновационная технология проектирования изделий и предметно-пространственной среды: инклюзивный аспект», 23 марта Москва 2019, Международном научно-техническом симпозиуме, Международного Косыгинского Форума 29 октября, Москва 2019, Всероссийской научной конференции молодых исследователей с международным участием, посвященной Юбилейному году в ФГБОУ ВО «РГУ им. А.Н. Косыгина»: Инновационное развитие техники и технологий в промышленности (ИНТЕКС-2020), 14 – 16 апреля, Москва 2020, Международной научно-технической конференции: «Дизайн, технологии и инновации в текстильной и легкой промышленности (ИННОВАЦИИ-2020)» 12 ноября, Москва 2020,

Международной конференции: Концепции, теория, методики фундаментальных и прикладных научных исследований в области инклюзивного дизайна и технологий 25-27 марта Москва, 2020, 72-ой Внутривузовской научной студенческой конференции «Молодые ученые –инновационному развитию общества (МИР-2020)», посвященной юбилейному году в РГУ им. А.Н. Косыгина (16-20 марта), Москва, 2020, Международной научно-практической конференции «Фундаментальные и прикладные научные исследования в области инклюзивного дизайна и технологий: опыт, практика и перспективы» (24 – 26 марта 2021 г.), Москва, 2021. Международном научно-техническом симпозиуме «Современные инженерные проблемы в производстве товаров народного потребления» III Международного Косыгинского Форума «Современные задачи инженерных наук» (20-21 октября 2021 года). Получено Свидетельство Федеральной службы по интеллектуальной собственности о государственной регистрации программы для ЭВМ «Вычислительный сервис облачной системы автоматизированного проектирования обуви Calceus» № 2021615416 (Дата публикации: 07.04.2021) (Приложение В). Дипломы ряда конференций различного уровня свидетельствуют о состоятельности предлагаемых в диссертации решений.

Публикации. Основные положения диссертации отражены в 14 печатных работах, 3 из которых опубликованы в ведущих рецензируемых изданиях, рекомендованных ВАК.

Структура и объем работы. По своей структуре диссертация состоит из введения, 3-х глав, выводов по главам и работе в целом, списка литературы, приложений. Работа изложена на 184 страницах машинописного текста, включает 34 рисунка, 2 таблицы. Список использованной литературы содержит 228 наименований библиографических и электронных источников. Приложения представлены на 6 страницах.

1. ВОЗМОЖНОСТИ ОБЛАЧНЫХ ТЕХНОЛОГИЙ ПРИМЕНИТЕЛЬНО К САПР ОБУВИ

1.1 Пандемия и работа в режиме удаленного доступа

В связи с пандемией в стране и в мире в середине марта 2020 г. в 26 регионах Российской Федерации Правительство для обеспечения безопасности граждан ввело режим самоизоляции сначала на 14 дней, а затем неоднократно переносило сроки его окончания. В этот период огромное количество человек потеряли работу, закрылось много предприятий, особенно малого и среднего бизнеса, индивидуальных предпринимателей. По данным Единого государственного реестра налогоплательщиков (ЕГРН), количество закрывшихся или заявивших о ликвидации предприятий малого и среднего бизнеса в этот период составило более 1.5 млн. юридических лиц, а индивидуальных предпринимателей – более 800 тыс. человек [11].

Работодатели и работники столкнулись с проблемами, требующими принятия быстрых и нестандартных решений, кардинальным образом повлиявших на трудовые отношения. Работодатели оперативно перевели своих сотрудников на домашний режим работы. Поскольку трудовое законодательство не позволяло легально оформлять такой перевод, возникло множество сложностей, связанных с регламентом рабочего времени и времени отдыха, использованием информационно-коммуникационных технологий, оплатой. Легче всего перевод на удаленный режим работы осуществлялся компаниями, в которых был введен электронный документооборот и приняты надлежащие локальные нормативные акты [12].

Институт мировой экономики Петерсона рекомендует учитывать сценарий «длительного COVID», так как неизвестно, когда мир перейдет в

постпандемическую фазу. Наиболее вероятным исходом событий называют периодические волны заражения.

Ввиду этого, компании не спешат с отказом от удаленной работы. Исследование, проведенное «Делойтом» и X5 Group, показало, что до пандемии работа удаленно не представлялась возможной в 27% опрошенных компаний. Столько же компаний сообщили о неформальной договоренности с сотрудниками об удаленной работе, при ее формальном запрете. После начала пандемии, 97% опрошенных компаний позволили сотрудникам работать удаленно или в гибридном формате даже после снятия ограничений регулятора, 77% из этих компаний организовали выплаты компенсации за изменение формата работы, а 36% компаний оказывают материальную поддержку сотрудникам с детьми [13].

Опрос, проведенный Райффайзенбанком среди тысячи жителей крупных городов России, показал положительное отношение к удаленной работе 88% россиян. Среди основных преимуществ 78% опрошенных называют экономию времени на дорогу, 49% - существенное сокращение расходов. Столько же опрошенных отметили возможность проводить больше времени с семьей. 33% опрошенных сообщили, что в офисе они отвлекаются на шум и разговоры коллег, отсутствие этих факторов дома позволяет им работать эффективнее. За последний год более половины 54% опрошенных работали удаленно на постоянной основе, остальные – в гибридном формате [14].

Дистанционный режим работы возможен только в тех сферах деятельности, которые не требуют физического присутствия для обеспечения технологического процесса, а также имеют возможность использовать оборудование и программно-технические средства вне производства.

Например, такой сферой деятельности для обувных предприятий является конструкторская подготовка.

Запрос на дистанционную работу модельеров-конструкторов обуви существовал и до пандемии по следующим причинам:

- размещение производства в других странах приводит к необходимости командирования модельеров-конструкторов, что может стать препятствием для специалистов.
- начинающим компаниям довольно трудно содержать несколько рабочих мест и естественным выходом становится аутсорсинг – передача некоторых обязанностей на сторону.
- образовательная миграция – активный отток выпускников школ, стремящихся поступить в столичные вузы, и которые впоследствии становятся перспективными молодыми кадрами для центральных регионов России [15].

Одним из основных инструментов модельера-конструктора является система автоматизированного проектирования. Рассмотрим подробнее такие системы с позиции дистанционного-режима работы.

1.2 Современные системы автоматизированного проектирования обуви

Сегодня без системы автоматизированного проектирования не обходится ни одно производство. САПР относятся к 3-й промышленной революции, ознаменовавшей повсеместное применение информационно-коммуникационных технологий в производстве, что способствовало формированию постиндустриального общества [16]. Первые системы автоматизированного проектирования появились еще в 60-х годах. Хэнретти П., сотрудник компании GeneralMotors, разработал графическую систему

подготовки производства. Свою работу он продолжил в собственной компании Manufacturing and Consulting Services (MCS). Начиная с 1971 года началось формирование современного образа САПР [17]. К середине 80-х САПР обрели форму, которая существует и сейчас. Развитие аппаратных графических ускорителей позволило уже к началу 90-х годов обрабатывать трехмерную графику. Снижение стоимости компьютеров позволило выйти на рынок САПР новым компаниям, что усилило конкуренцию и стимулировало совершенствование продуктов [18].

Для проектирования обуви применяют специализированные САПР. В таблице 1 представлен список таких систем, полученный из анализа открытых источников. Существенным различием между ними являются следующие параметры:

- формат, в котором производится проектирование - 2D или 3D;
- техническая форма: одни системы являются самостоятельными программами, другие же - расширением существующих;
- активное развитие: некоторые системы хотя и прекратили свое развитие, но все еще применяются на производстве.

Таблица 1. Распространенные САПР обуви

Название	Тип	Активно развивается	Формат	Страна
KSM FootwearCAD	Программа	Нет	2D/3D	Польша
iCad3d+	Программа	Да	2D/3D	Испания
EliStile 3D	Программа	Да	3D	Италия
ElitronCAD 2D Footwear	Программа	Да	2D	Италия
MindCad	Программа	Да	2D/3D	Португалия
ShoeExpress	Плагин Cimatron	Да	2D/3D	Израиль

Romans CAD	Программа	Да	2D/3D	Франция
ShoeMaster	Программа	Да	2D/3D	Италия
PROCAM DIMENSIONS	Программа	Да	2D/3D	Австрия
Mirisys FootwearCAD	Программа	Да	2D	Чехия
ClassiCAD	Программа	Да	2D	Чехия
LeaCAD	Программа	Да	2D/3D	Италия
Naxos	Программа	Да	2D/3D	Италия
Pragma	Программа	Да	2D/3D	Италия
Visi shoes	Программа	Нет	2D/3D	Великобритания
USM 2	Программа	Да	2D	Великобритания
Caligola	Программа	Да	2D	Италия
Аско-2D	Программа	Да	2D	Россия
ShoesModel	Программа	Да	2D	Россия
Ассоль обувь	Программа	Да	2D/3D	Россия
ShoeMaker	Программа	Нет	2D/3D	Великобритания
ImagineElf	Плагин Rhino3D	Нет	3D	Китай
PatternElf	Плагин Rhino3D	Нет	2D	Китай
GradingElf	Плагин Rhino3D	Нет	2D	Китай
Jevero	Плагин Rhino3D	Да	2D/3D	Италия

Распространенность систем чаще всего ограничена страной компании-разработчика. Так, среди российских обувных предприятий популярны преимущественно отечественные системы автоматизированного проектирования формата 2D. Это обусловлено доступностью систем, а также

их достаточной гибкостью и способностью встраиваться в существующий рабочий процесс [19]. Наиболее распространенной системой является АСКО-2Д. В разных городах России ее применяют более 100 обувных предприятий. Помимо моделирования, система предоставляет возможность деталировки, градирования, оцифровки при помощи дигитайзера, расчета укладываемости деталей, создания конструкторской и технологической документации [20]. Менее распространенной является Ассоль-обувь (МФТИ), которая также, как и АСКО-2D, содержит функции деталировки, градирования, оцифровки при помощи сканера или дигитайзера, расчета укладываемости деталей, создания конструкторской документации [21].

Системы формата 3D не получили распространения в России из-за своей высокой стоимости и сложности в работе. Наиболее успешные попытки внедрения таких систем были у компании Crispin [19]. Их комплекс включал следующий набор программ:

- LastMaker – моделирование колодки;
- ShoeMaker – конструирование деталей верха в формате 3D;
- SoleEngineer – конструирование деталей низа обуви;
- EngineerPro – конструирования деталей обуви в формате 2D;
- PatternCut – получения схемы раскроя;
- OrthoModel – моделирование ортопедических стелек;
- OrthoMill – формирование технологического процесса производства смоделированных в OrthoModel стелек;
- ShoeCost – расчета стоимости обуви;
- TechPac – создание и ведение различного рода документации;

Однако этот комплекс программ прекратил существование: в 2007 году, компания Crispin стала частью компании Delcam – производителем САПР технологической подготовки; в 2014 году Delcam была куплена компанией

Autodesk и к 2018 году компания Autodesk полностью закрыла разработку и продажу всех программ Crispin и Delcam, кроме основных – PowerSHAPE, PowerMILL и FeatureCAM [22].

Рассмотрим процесс проектирования в системах АСКО-2D и Shoemaker, чтобы выявить наиболее востребованные функции систем.

Перед тем как начать работу в системе, разрабатывается эскиз модели обуви и подбирается колодка. Модельер, например, по итальянской методике, наносит на предварительно обклеенную скотчем колодку, основные и вспомогательные линии модели обуви, и получает развертку – грунд-модель. После этого в САПР создается новый проект, вводится общая информация о модели обуви, грунд-модель оцифровывается при помощи дигитайзера. Затем, оцифрованный чертеж корректируется и начинается процесс построения различных элементов: блочек, припусков, сборочных линий и гофр. После этого в САПР вводится информация о деталях, с указанием принадлежащих ей линий и элементов – деталировка. Производится градирование деталей на заданный размерный ряд. Затем проводят раскладку деталей на листе для вывода на печать. При помощи плоттера печатают или вырезают детали. Также на печать может выводиться сопроводительная документация – информация о модели и ее деталях [23]. Таким образом, основные этапы инженерно-технического художественного конструирования выполняются вне САПР, а автоматизация затрагивает только самые трудоемкие задачи.

Системы формата 3D предлагают модельеру работать по итальянской методике с трехмерной моделью колодки. Алгоритм автоматического получения развертки осуществляет переход в подсистему формата 2D и далее процесс проектирования идентичен САПР формата 2D [23]. Однако, несмотря на функциональность, системы не навязывают модельеру такой метод

проектирования, предлагая функции оцифровки при помощи дигитайзера. Таким образом, системы формата 3D функционально являются своего рода дополнением к системам формата 2D. На наш взгляд, общая схема функционирования САПР, которая отражает этапы работ, характерных как для систем формата 2D, так и для систем формата 3D, может быть приведена, к виду, представленному на рисунке 1.1.

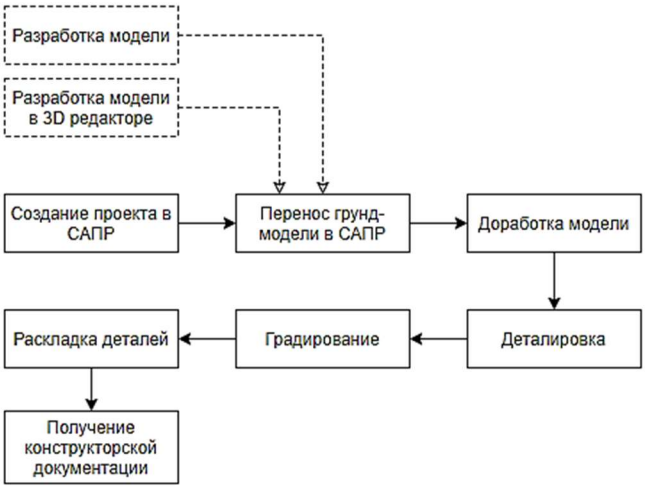


Рисунок 1.1 Этапы работ в САПР обуви

Приведем более подробный список функциональных возможностей, характерных для большинства САПР обуви.

1. Моделирование в формате 2D – работа с чертежом конструктивной основы верха обуви. Эта функциональность является основой САПР. Если система не имеет такой возможности и, например, работает только в 3D, то она обязательно имеет функцию выгрузки модели в формат 2D.
   1.1. Оцифровка чертежа при помощи внешнего устройства – дигитайзеры или сканеры ускоряют операцию переноса чертежа в САПР.
   1.2. Импорт и экспорт модели в файлы открытых форматов – использование форматов DWG, SVG и других позволяет производить обмен между различными информационными системами.

1.3. Деталировка модели – функция разбора модели на детали.

1.4. Градирование деталей – функция масштабирования деталей на размерный ряд.

1.5. Получение схемы раскроя деталей – информация об эффективной укладываемости деталей различными методами.

2. Моделирование в формате 3D – работа представляет из себя нанесение базисных линий на виртуальную колодку и автоматическое получение развертки модели.

2.1. Моделирование колодки – возможность редактировать и создавать новые виртуальные колодки.

2.1.1. Оцифровка колодки при помощи внешнего устройства – функция переноса реальной колодки в САПР. Чаще всего внешним устройством является 3D сканер. Несовершенство устройств оцифровки обязывает систему иметь возможность моделирования колодки для устранения погрешностей.

2.2. Импорт и экспорт колодки – позволяет моделировать колодку в сторонних программах. Возможность импорта из открытых форматов IGES, STL, OBJ и других позволяет применять внешнее устройство оцифровки, независимо от поддержки таких устройств самой САПР.

2.3. Импорт и экспорт моделируемой обуви – дает возможность проектирования в сторонних программах трехмерного моделирования. САПР в таком случае используется только для получения развертки трехмерной модели. Ввиду сложности структур данных трехмерных САПР, чаще всего импорт не реализован, в отличии от экспорта, который используется для визуализации обуви в сторонних программах.

2.4. Моделирование вторичных объектов – возможность создавать и редактировать различные объекты, применяемые на этапе визуализации – шнурки, застежки и прочее.

2.5. Импорт вторичных объектов – позволяет моделировать вторичные объекты в сторонних программах трехмерного моделирования или получить реальные объекты при помощи 3D сканера.

2.6. Рендеринг (визуализация) обуви – возможность получения цифрового растрового изображения 3D модели.

2.6.1. Применение шейдеров при рендеринге –технология получения фотореалистичных изображений.

Анализ технических свойств перечисленных систем позволил выявить недостатки, влияющие на эффективную организацию дистанционного режима работы.

Стоимость. При выборе системы автоматизированного проектирования, ее цена является одним из решающих факторов [19].

Разработчики крупных САПР общего назначения для снижения стоимости перешли в формат временной подписки, которая включает собственно программу, техническую поддержку и иногда курс обучения [24]. Среди обувных САПР чаще встречается классическая «пожизненная» модель лицензирования – система приобретается один раз и навсегда, в стоимость также входит поддержка на срок от одного года [21,25]. Цена зарубежных систем достигает отметки в 10000 евро [26], отечественных – от 200000 до 300000 рублей [27]. Покупка таких систем может быть проблематичной для малых обувных предприятий, поэтому чаще они применяют не САПР, а векторные графические редакторы Adobe Illustrator или Corel Draw, реже AutoCAD [26,28].

Защита программного обеспечения. Решая проблему «компьютерного пиратства» – нелегального копирования и распространения программ, разработчики создали множество систем защиты. Изначально это были только ключи в виде набора символов, уникальные для каждой копии программы. Проверки на то, что активирована одна копия на один ключ не было, поэтому множество нелегальных копий работали с одним и тем же ключом. Затем были придуманы «привязки» лицензионного ключа к уникальным характеристикам компьютера, к CD или DVD диску. С развитием интернета распространились проверки лицензионного ключа через сеть [29]. Вершиной развития систем защиты стал аппаратный ключ – устройство, которое присоединяется к компьютеру, чаще всего через порт USB. Защищенная программа частично зашифрована и не способна выполняться без полной расшифровки, которую и выполняет присоединенный аппаратный ключ [30]. Различные профессиональные программы применяют его для своей защиты. Например, программы автоматизации бизнеса 1С или отечественные САПР обуви: АСКО-2D, Ассоль-обувь, ShoesModel [20,21,31]. С точки зрения пользователя работать с аппаратным ключом проблематично. Устройство размером с флэш-накопитель легко потерять, что равносильно покупке новой копии программы. Так, дистрибьюторы 1С восстанавливают ключ только в двух случаях – при краже и пожаре. Для этого необходимо предоставить справку из полиции о возбуждении уголовного дела или справку из МЧС о факте пожара [32].

Неэффективная оцифровка. Инженеру-конструктору необходимо перейти от объемной формы колодки к плоской форме деталей, т.е. получить их развертку – чертеж и перенести его в САПР обуви [33]. Традиционный способ ввода чертежа в компьютер предполагает использование дигитайзера, при помощи которого оператор последовательно указывает точки контура чертежа на достаточно близком друг к другу расстоянии для высокой точности

оцифровки. Отдельные элементы – надсечки, оси симметрии вводятся тем же способом. В разных САПР процесс оцифровки организован с разной степенью удобства, в любом случае ручной ввод занимает значительное время конструктора и может привести к ошибкам [34]. Такой метод не подходит для организации дистанционного режима работы, так как необходимо приобрести дорогостоящий дигитайзер для каждого рабочего места. Альтернативный метод предполагает ввод не чертежа, а контуров лекал с помощью обработки изображения программой. Лекала разделяют на первичные и вторичные. Изображения первичных лекал вводят в программу, где векторизуют контуры и на их основе достраивают вторичные лекала (оформляют углы, надсечки, строят припуски, вводят спецификации лекал). Для получения изображения лекал используются цифровые камеры и сканеры [34]. Такой метод более предпочтителен для дистанционной работы, так как использует более дешевые устройства, чем дигитайзер. Однако введение контуров отдельных лекал, а не всего чертежа подразумевает, что часть работы, которую могла бы автоматизировать САПР, должна быть проделана вручную.

Описанные проблемы можно решить, используя облачные технологии. Рассмотрим их подробнее.

1.3 Текущее состояние применения облачных технологий в САПР

Применение облачных технологий в контексте цифровизации производства описано в концепции «Индустрии 4.0», как инструмент обновления производственной системы для поддержки ее конкурентоспособности [35]. Изначально парадигма «Индустрии 4.0» была сфокусирована на «умном производстве» - объединение физической и цифровой среды. Но последствия первой волны внедрения цифровых

технологий в производство показали, что этот фабрично-ориентированный подход слишком прост, и интернет несет гораздо больший потенциал. Цифровые сети могут создать интегрированную систему участников, где в реальном времени будет происходить обмен данными между поставщиками, фабрикой, дистрибьюторами и клиентами [36].

Исследователи Тёниссен Х. и Бертола П. в своей работе «Innovating Fashion Industry Through Digital Transformation» [37] представили графическое описание «Индустрии 4.0», приведенное на рисунке 1.2.

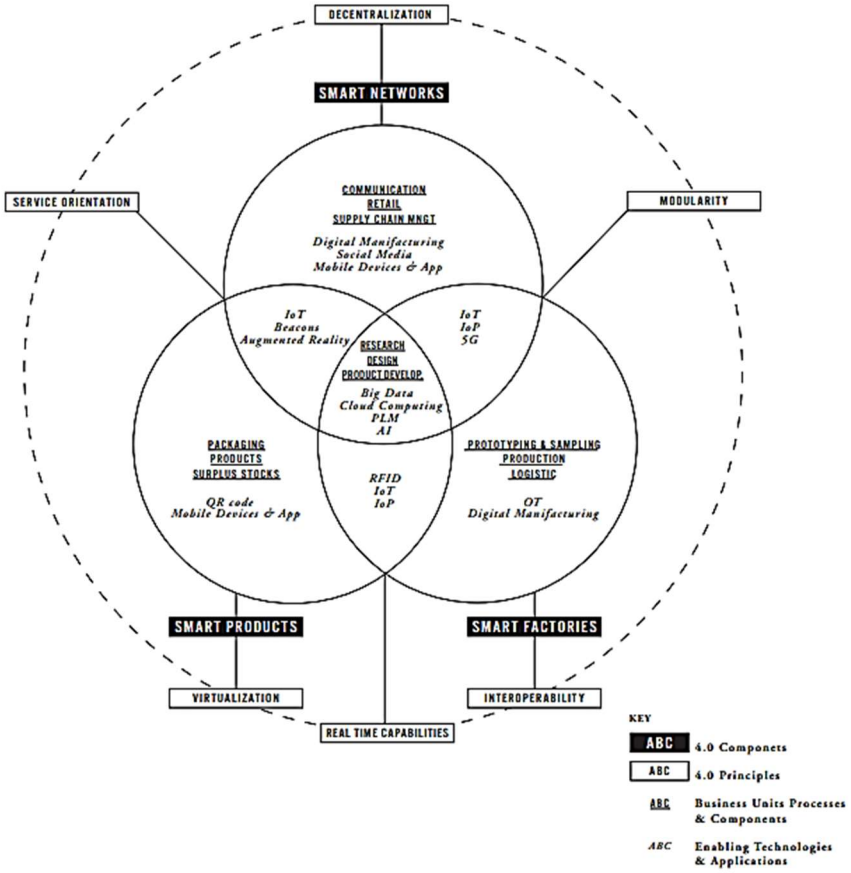


Рисунок 1.2 Компоненты и принципы индустрии моды 4.0

Центром этой схемы являются процессы исследования, дизайна, разработки. К технологиям, реализующим эти процессы, причислены облачные технологии и системы автоматизированного проектирования.

Национальный институт стандартов и технологий (NIST) дал наиболее полное определение облачным технологиям – это модель предоставления удобного сетевого доступа в режиме «по требованию» к коллективно используемому набору настраиваемых вычислительных ресурсов (например, сетей, серверов, хранилищ данных, приложений и/или сервисов), которые пользователь может оперативно задействовать по свои задачи и высвобождать, при сведении к минимуму числа взаимодействий с поставщиком услуги или собственных управленческих усилий [38]. Приведем основные характеристики облачных технологий [39].

Самообслуживание по требованию. По мере необходимости, пользователь самостоятельно конфигурирует вычислительные ресурсы, в автоматическом режиме через панель управления облачной платформы.

Доступность через сеть. Доступ к ресурсам осуществляется при помощи стандартных протоколов передачи данных, например HTTP. Это позволяет облачным системам взаимодействовать со множеством различных видов клиентский устройств.

Объединение ресурсов в пул. Поставщик объединяет несколько физических серверов и предоставляет к ним доступ, как к одной вычислительной машине. В соответствии с потреблением вычислительных мощностей, ресурсы динамически распределяются по физическим серверам. Заказчик способен контролировать местоположение серверов только на абстрактном уровне – страна, регион, дата-центр.

Способность к адаптации. Вычислительные ресурсы способны быстро и гибко резервироваться для оперативного масштабирования под задачи пользователя, и также быстро освобождаться.

Измеряемые ресурсы. Облачные системы автоматически контролируют и оптимизируют использование ресурсов через измерение определенных параметров. Эти параметры варьируются в зависимости от типа услуги. Например, это могут быть: размер хранилища данных, вычислительная мощность, пропускная способность и/или число активных пользовательских записей. Использование ресурсов отслеживается, контролируется; формируются отчеты. Таким образом и поставщик, и пользователь получают прозрачную информацию об объеме оказанных услуг.

Концепция облачных технологий получила распространение относительно недавно, но зарождение происходило еще в 1950-х годах. Тогда вычислительные машины имели высокую стоимость, и компании не могли приобрести их для большого числа сотрудников. Была предложена концепция разделения времени, предполагающая подключение нескольких рабочих станций к одной вычислительной машине. Реализация этой идеи началась в 1959-м году, а первое коммерчески успешное решение выпустили в 1964-м году [40]. Ввод и вывод данных производился посредством работы с терминалами, а вычисления производились на сервере. Терминалы обладали минимальным необходимым количеством вычислительных мощностей для ввода-вывода [41]. В начале 1970-х, Ликлайдер Д. предложил из вычислительных центров создать глобальную сеть, которая позволяла бы получить доступ к вычислительным устройствам из любой точки мира [40]. А Д. Маккарти высказал идею о вычислениях, как услуге, которые бы выполнялись с помощью «общественных вспомогательных средств» [41]. Модель утратила свою актуальность в 1980-х годах, с появлением дешевых персональных компьютеров [40].

Первым и самым крупным поставщиком облачных услуг является Amazon, открывший в 2005-м году онлайн хранилище данных Simple Storage Service (S3). Упор в развитии облачных технологий Amazon сделали на

предоставление аппаратных мощностей, создав платформу Amazon Web Services (AWS). Это направление сформировало конечное представление о модели предоставления вычислительных мощностей. Сегодня Amazon занимает 33% рынка. Компания непрерывно ведет экспансию, вкладывая средства в открытие новых центров обработки данных по всему миру [41]. Следом за Amazon по масштабности идет Microsoft. Их облачная платформа Azure появилась только в 2010-м году. Ее руководитель разработки Сатья Наделла стал главой Microsoft в 2014-м году, и кардинально сменил вектор развития компании в сторону облачных технологий. Преимущество Microsoft было в экосистеме, которую компания развивала для бизнеса с 90-х годов: серверная операционная система Windows Server, система управления базами данных SQL Server, служба каталогов Active Directory, веб-сервер IIS, система виртуализации Hyper-V [42], программная платформа DotNet, и прочие. Возможность этих инструментов гармонично взаимодействовать друг с другом позволяет разработчикам сократить время создания программ в разы. А их применение в облачных технологиях сформировала конечное представление о модели предоставления готовых облачных решений, сделав Microsoft лидером в этой сфере [41]. Последующий перенос множества бизнес-решений в облако, например Microsoft Office, сделало Microsoft лидером в этой области. Третьей по величине облачной платформой является Google Cloud Platform (GCP). Она была запущена позже AWS и Azure, в 2011 году. Платформа Google не имеет масштабов Amazon, и экосистемы, сравнимой с Microsoft. Однако GCP занимает уверенное третье место на рынке поставщиков облачных вычислений. Это обусловлено тем, что Google первыми стали предлагать услуги размещения контейнеров в облаке, и их оркестрации при помощи системы Kubernetes [42]. В дополнение Google открыла для использования в облаке свои собственные

разработки в области искусственного интеллекта и обработки больших данных. Таким образом компания нашла свою нишу в сегменте облачных технологий.

В России так же существуют компании, предоставляющие услуги облачных сервисов. С 2017 года было произведено больше количество запусков платформ в сегменте интернет-компаний и операторов сотовой связи. Самые крупные из них: Mail.Ru, Яндекс, Сбер, МТС, Мегафон, Вымпелком, Ростелеком, Softline, Dataline, КРОК, Selectel. Компании предоставляют все модели обслуживания облачных систем и ориентированы на упрощенную миграцию. Платформы используют распространенные системы с открытым исходным кодом, что упрощает создание новых или переноса существующих систем на платформу – миграцию. Собственные системы также ориентированы на миграцию, например, облачное хранилище Mail Cloud Solutions полностью совместимо с S3 – хранилищем Amazon AWS. Платформа МТС построена на системе AzureStack, делая ее полностью совместимой с облачной платформой Microsoft Azure. Также компании предлагают: платформы для хранения и обработки больших данных; сервисы для разработки приложений на основе машинного обучения; платформы интернета вещей; собственные разработки в области распознавания речи и изображения; балансировщики нагрузки; межсетевые экраны, как общие так и защиту от внешних атак; виртуальные сети; соединение с локальной инфраструктурой; сети доставки контента; пакетную, событийную и потоковую обработку данных; средства разработки и тестирования программного обеспечения. Также, каждая платформа имеет магазин с готовыми решениями для бизнеса: ERP и CRM системы, модули интернет-магазинов, электронная почта, системы документооборота и так далее. Отечественные платформы выгодно отличаются от зарубежных наличием аттестата ФСТЭК России, гарантирующего соответствие ФЗ-152 «О защите персональных данных» [41].

Развитие облачных технологий было затруднено следующими убеждениями [43].

Монополия поставщика. Интеграция новой программной системы в компанию несет большие риски, это всегда многоэтапная и сложная задача, увеличивающая финансовую нагрузку на предприятие. Для малого бизнеса подобный переход может привести к большим потерям. Однако, такая ситуация была актуальна, когда облачные технологии находились на начальном этапе развития, новые разработчики экспериментировали и осваивали рынок SaaS, в то время как создатели зарекомендовавших себя систем не спешили переносить свою кодовую базу в облако, из-за отсутствия такого опыта. Сегодня, большинство крупных компаний, системами которых организации пользовались много лет, уже перенесли свои решения в облако [44]. Примерами такого переноса являются: «1С: Предприятие», «Битрикс24», Microsoft Office 365, «СКБ Контур», и другие.

Неконтролируемые изменения. В современном мире полезность применения программных средств во многом зависит от скорости внедрения новых функций. Существуют опасения, о различном представлении необходимых доработок у пользователя и у поставщика. Это говорит о том, что пользователь не видит разницы между массовой системой и системой, выполненной на заказ. Десктопное программное обеспечение при использовании и обслуживании не отличается от облачного, их нужно обновлять по мере нахождения ошибок и уязвимостей. Поэтому недовольство пользователей от изменений в очередном обновлении не зависит от того на каком сервере расположена система. При создании программного обеспечения на заказ, с клиентом взаимодействует определенный круг лиц для определения требуемого функционала, и дальнейшей стратегии развития. Как правило, в IT-

компаниях такие должности называются «владелец продукта» (product owner) и «менеджер продукта» (product manager) [45].

Стандартный набор функций. Разработчики программных средств реализуют некоторые общие бизнес-процессы, что может быть преимуществом для начинающей организации. Но для компании, обладающей выстроенными уникальными процессами работы, внедрение необходимых ей функций может оказаться затруднительным. Так же, как и в предыдущем случае, внедрение новых функций по требованию пользователя возможно только при создании систем на заказ, независимо от того десктопная эта система или облачная [45].

Негарантированный доступ. Для доступа пользователя к облачным ресурсам необходим интернет.У интернет-провайдеров могут возникать как программные сбои сетевых маршрутизаторов, так и обрывы проводной связи, на восстановление которых может потребоваться несколько суток. Все это время пользователь не может работать с облачными сервисами. Отсутствие интернета и плохая связь – единственная серьезная проблема, при которой облачные технологии использоваться не должны. Поставщики услуг это понимают и зачастую предлагают пробный период, чтобы клиент мог оценить работоспособность облачного решения. В свою очередь, от перебоев никто не застрахован, они могут случаться не только со связью, но и с электричеством, однако предприятия не спешат покупать альтернативные источники электроэнергии на случай ее отключения [46].

Конфиденциальность. Кибербезопасность — основная задача интернет-сообщества. Поставщики облачных сервисов, как потенциальные хранители данных множества пользователей, будут более привлекательны для атак, чем локальные сети организаций. Что справедливо для малых предприятий, при расширении которых возникнут задачи: сегментации сети, внедрения

межсетевого экрана, системы предотвращения вторжений (IPS), системы управления информацией о безопасности (SIEM), и, вероятно, внедрения корпоративной виртуальной частной сети (VPN). Компаниям придется создавать отдел информационной безопасности, который должен создать инфраструктуру и затем поддерживать ее в рабочем состоянии. В дополнение к этому, необходимо ежегодно проводить аудит безопасности сторонними организациями. Поставщики облачных технологий, в свою очередь, проводят все эти мероприятия для обслуживания всех клиентов одновременно, ввиду того что они находятся в одной сети. Это одна из причин экономичности облачных услуг [47].

Сохранность. Центры обработки данных, как и любые другие здания и сооружения, подвержены стихийным бедствиям. В августе 2015 года молния 4 раза ударила по дата-центру Google в Бельгии. Часть данных пользователей была безвозвратно утеряна. По сообщению Google пострадала 0,000001% данных, принадлежавших Google Compute Engine (GCE) – это сервис, предоставляющий пустые виртуальные сервера. Таким образом, пострадала инфраструктура, которую пользователи выстраивали самостоятельно на виртуальных машинах Google. В собственной инфраструктуре можно развернуть базу данных, однако ответственность за ее данные несет пользователь. Поставщик облачных технологий не настраивает и не контролирует компоненты, которые пользователь самостоятельно размещает на виртуальных серверах. Поставщики облачных технологий, рекомендуют, вместо развертывания собственных, использовать их готовые инфраструктурные решения. Например, базы данных Google не пострадали от удара молнией, так как у них настроено регулярное резервное копирование данных [48].

Несмотря на возможные проблемы, разработчики систем автоматизированного проектирования общего назначения постепенно внедряют облачные технологии.

Autodesk – американская компания, разрабатывающая системы моделирования общей направленности, а также в области архитектуры и медиа. Основными программными решениями являются: САПР AutoCAD, система информационного моделирования зданий и сооружений Revit, комплексная САПР конструкторской и технологической подготовки Fusion 360 (рис.1.3), системы полигонального трехмерного моделирования 3DS Max и Maya.

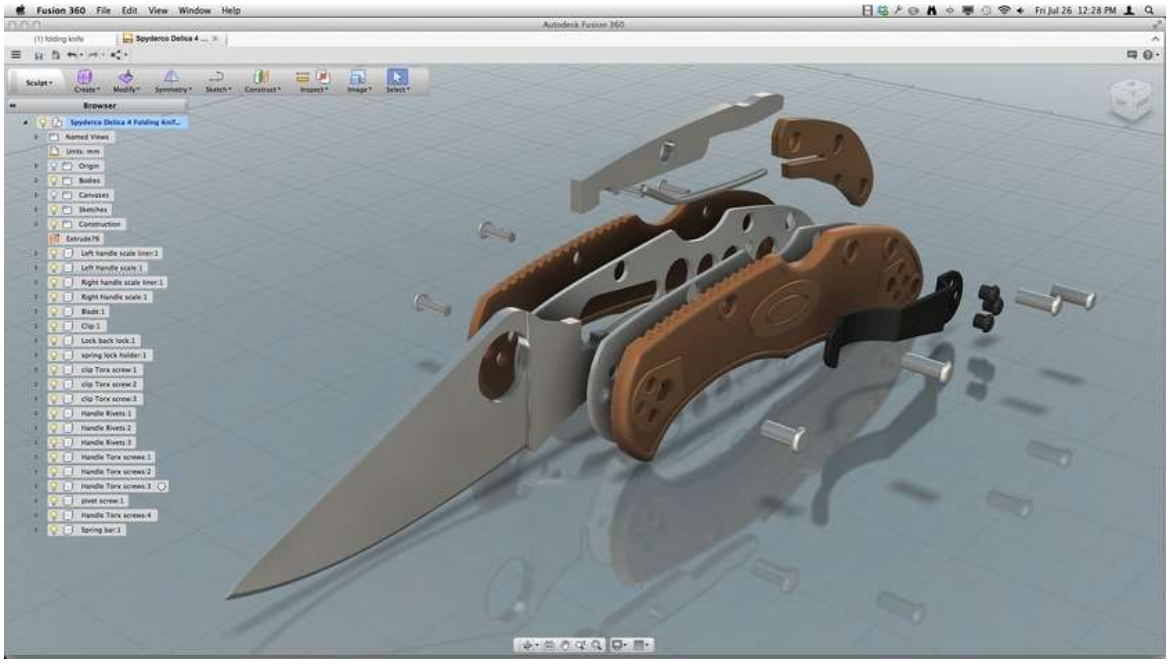


Рисунок 1.3 Облачная САПР Autodesk Fusion 360

Внедрение облачных технологий Autodesk начала еще в 2012 году. Облачная платформа Autodesk включает 2 системы: персональную A360 Drive и групповую A360. Система A360 Drive представляет собой облачное хранилище с системой контроля версий проектных данных. Хранилище предоставляется как предприятиям, так и персонально для каждого пользователя. Вторая система A360 - сосредоточена на тесном взаимодействии

пользователей. Групповая разработка предполагает создание проекта, предоставление доступа к нему отдельным пользователям, а также распределение ролей. Таким образом, компания Autodesk рассматривает облачные технологии как инструмент совместного проектирования [49].

Siemens – немецкий конгломерат, разрабатывающий множество технических решений в различных отраслях, основными программными решениями является система полного цикла разработки и проектирования Siemens NX (рис. 1.4), а также САПР общего назначения Siemens Solid Edge.

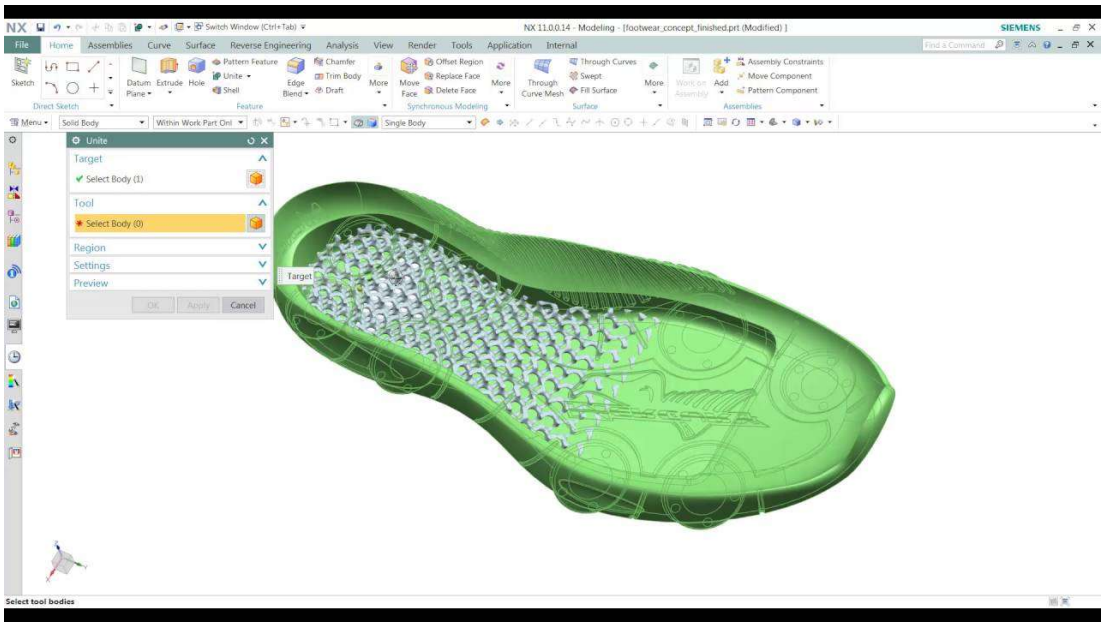


Рисунок 1.4 Облачная САПР Siemens NX

Компания выделяет несколько причин внедрения облачных технологий в свои системы: быстрое внедрение систем, отсутствие необходимости предприятиям управлять информационными ресурсами, экономия средств и снижение рисков, поддержка удаленного доступа к проекту, применение неограниченных вычислительные мощностей для уникальных возможностей (например, генеративное проектирование). Основной задачей облачных технологий, по мнению компании, является сосредоточение внимания на

инновациях и стратегическое развитие предприятия в области цифровизации при помощи цифровых двойников и интернета вещей [50].

Dassault Systemes – французская компания, создающая системы автоматизированного проектирования и управления жизненным циклом изделий для различных отраслей, включая комплексную систему полного цикла разработки CATIA и САПР общего назначения SolidWorks (рис. 1.5).

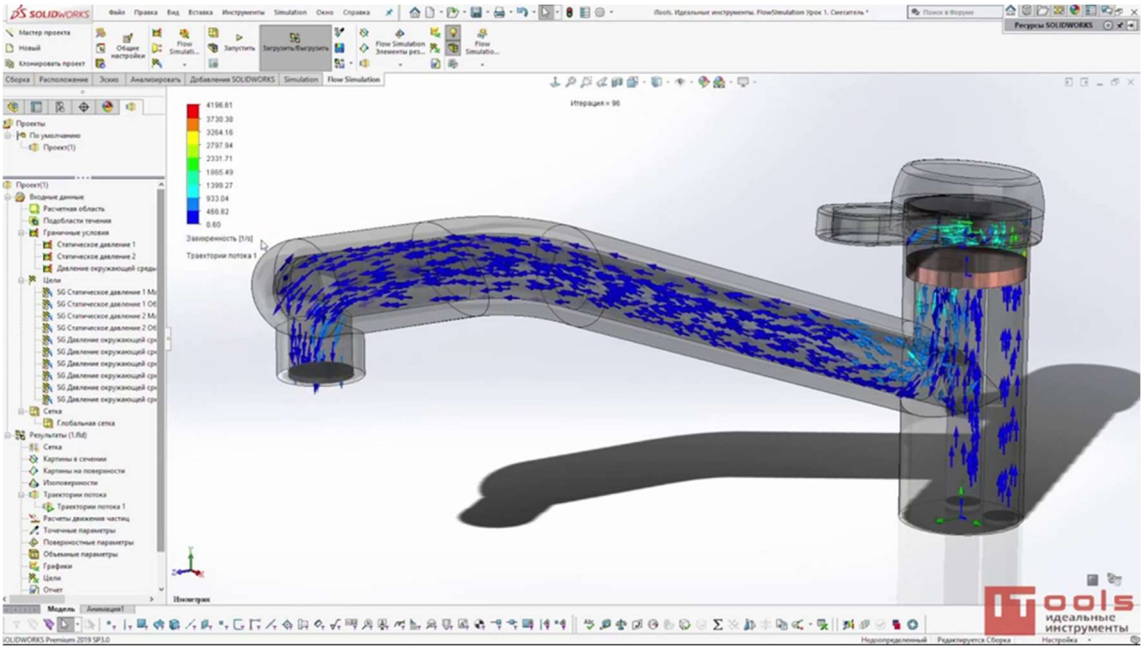


Рисунок. 1.5 Облачная САПР SolidWorks

Компания объединила все свои системы в единую облачную платформу 3DEXPIRIENCE. Главная проблема САПР, по мнению компании, является обновление программного обеспечения, что затрудняет обмен данными и совместную работу над новыми проектами. Переход на общую облачную платформу решил эти проблемы, создавав единую экосистему, в которой все заинтересованные стороны могут работать, когда и где угодно. Дизайнеры, производители, отделы продаж и маркетинга, и даже клиенты могут легко получить доступ к продукту на каждом этапе его разработки, в свою очередь тесное сотрудничество открывает доступ к инновациям. Другой положительной

чертой компания называет масштабирование и экономию. Облачные технологии рассчитаны на динамическое выделение вычислительных мощностей для удовлетворения потребностей проекта. Это помогает ускорить вывод продукта на рынок, повышая конкурентоспособность фирмы. А так как программные системы обновляются поставщиками, фирмам не нужно вкладывать значительные средства в локальные обновления системы, что особенно сложно в нынешних условиях [51].

Настоящим прорывом стала совершенно новая САПР OnShape, появившаяся в 2015-м году. OnShape – компания, выпускающая одноименную систему автоматизированного проектирования конструкторской подготовки, и состоящая из выходцев Dassault Systemes, а именно разработчиков SolidWorks. Система OnShape полностью построена на веб-технологиях, что характеризует систему, как Cloud-Native, т.е. программа создана специально для работы в облаке. Доступ к системе осуществляется через браузер, как показано на рисунке 1.6, решая вопрос кроссплатформенности.

Это выгодно отличает систему от ведущих САПР, которые попадают под понятие Cloud-Hosted – обычные программы, расширенные возможностями облачных технологий. На своем примере САПР OnShape показала на что способны веб-технологии сегодня, и какой может быть облачная система проектирования [52]. Инициатива была подхвачена компанией Autodesk, и уже в 2017-м году Fusion 360 стал доступен через браузеры в режиме тестирования, имея меньший функционал, чем оригинальная программа.

Однако, из-за высокой конкуренции OnShape не получила широкого распространения, хотя и собрала свою собственную аудиторию. По итогу система была куплена компанией PTC в 2020-м году.

Президент исследовательской компании Tech-Clarity Джим Браун прокомментировал это следующим образом: «Приобретение Onshape дает PTC стратегическое преимущество для удовлетворения растущего спроса на «САПР в облаке» и предоставляет компании чистую платформу для инновационных продуктов» [53].

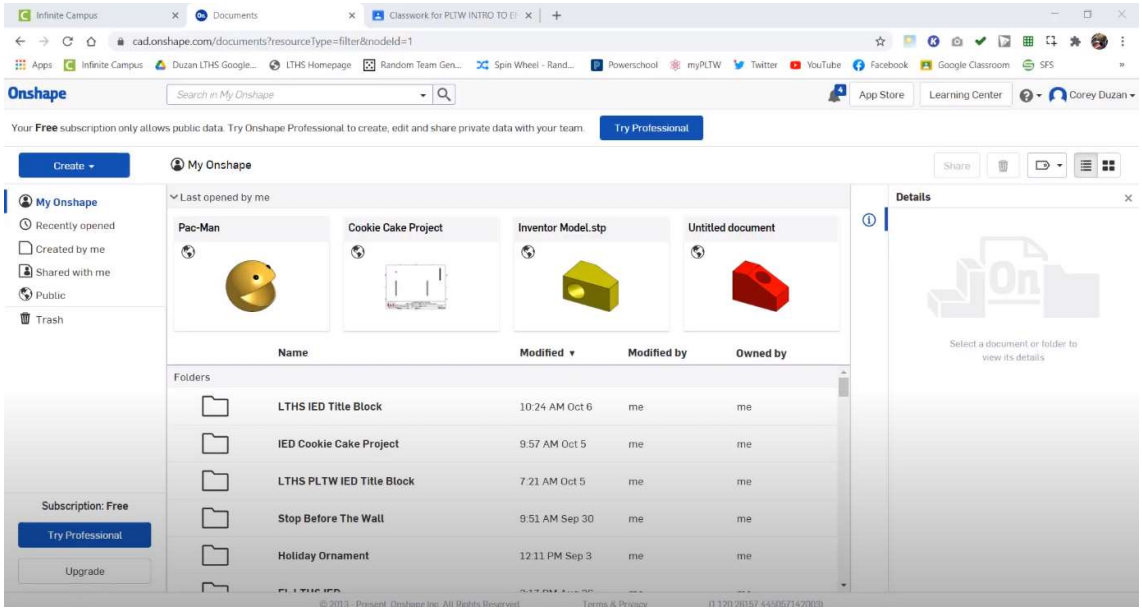


Рисунок 1.6 Облачная САПР OnShape, запущенная в браузере Google Chrome

Существующие облачные САПР обладают множеством функциональных возможностей. Однако их применение в производстве обуви затруднительно, ввиду отсутствия возможности оцифровки. Это обусловлено тем, что САПР общего назначения больше ориентированы под область машиностроения, в которой весь спектр работ производится в САПР [54]. Также стоит отметить, что методы проектирования в САПР общего назначения сильно отличаются от САПР обуви, так как предполагает построение в 3-х проекциях твердотельной модели из примитивов.

Анализ облачной системы OnShape, полностью построенной на веб-технологиях показал, что такой подход может решить проблемы организации

дистанционного режима работы модельера-конструктора обуви. Веб-технологии позволяют обеспечить доступ к системе с любого устройства из любой точки мира. Такая САПР более экономична ввиду того, что ее разработчикам нет необходимости обслуживать каждое пользовательское устройство. Конечный пользователь имеет возможность использовать только определенную часть системы, не оплачивая неиспользуемые ресурсы. Дистрибутив такой САПР размещается вне пользовательских устройств и не требует какой-либо специальной защиты в виде аппаратного ключа. Помимо этого, применение веб-технологий для создания САПР решают множество других проблем, характерных для десктопных программ [55,56].

Кроссплатформенность. Это возможность запуска программы на множестве различных платформ [57]. Под платформой понимается некоторая система, которая формирует среду запуска программного обеспечения [58]. Первые платформы были аппаратные, например, архитектура процессора, определяющая набор команд выполнения и систему адресации памяти. В общем случае, программа, скомпилированная под конкретный процессор, не может быть запущена на устройстве с процессором другой архитектуры [57]. Платформами являются и собственно устройства. Множество устройств может иметь компоненты одного типа, но от разных производителей, соответственно они будут иметь разные интерфейсы доступа к этим компонентам. Одна из задач операционных систем – предоставить унифицированный, независимый от производителя, доступ к компонентам устройства [59]. Однако со временем число различных операционных систем выросло, и они вошли в область понятия платформы. Для запуска программы на множестве операционных систем, независимо от их особенностей и интерфейсов доступа к компонентам, была создана среда Java [60]. Она состоит из трех элементов: специальный байт-код, независимый от архитектуры процессора; язык программирования,

который компилируется в данный байт-код; и программа, исполняющая команды байт-кода, которая берет на себя взаимодействие с операционной системой и устройством. Таким образом, программа, написанная один раз на Java, может запускаться на различных операционных системах и устройствах без каких-либо изменений. Спустя время была выпущена аналогичная среда DotNet от Microsoft [61], а затем большое распространение получили интерпретируемые языки программирования общего назначения [62], например, Python, которые не используют промежуточный байт-код. Программы-интерпретаторы разбирают исходный код и сразу же приступают к его выполнению. Таким образом появился еще один вид платформ. Серьезным недостатком такого рода платформ является невысокая производительность [63], поэтому большинство САПР, включая обувные, до сих пор создаются под конкретную операционную систему, чаще всего Windows [64]. Сегодня эта проблема стоит наиболее остро. В рамках импортозамещения программного обеспечения, российские разработчики создали операционные системы на основе ядра Linux [65], которое кардинально отличается от Windows, что делает невозможным применение распространенных САПР на отечественных операционных системах [66]. Облачные САПР поставляются посредством веб-технологий. Пользователь работает с системой через веб-браузер, который сегодня предустановлен на всех распространенных операционных системах, что является решением проблемы кроссплатформенности.

Развертывание и обновление. Сложные информационные системы, такие как САПР, практически невозможно создать без применения сторонних библиотек и прочих компонентов, именуемых зависимостями [67]. Например, применение программных платформ для решения проблемы кроссплатформенности уже создает зависимость от этой платформы. Также зависимостями являются «runtime библиотеки», которые содержат реализацию

стандартных функций языка программирования [68]. При этом бывает, что такие зависимости имеют свои собственные зависимости, создавая цепочки требований к системе. Проблемы, возникающие на этапе установки зависимостей, не всегда могут быть разрешимы разработчиком основной системы [69]. Аналогичная ситуация происходит и с процессом обновления. Веб-технологии, используемые облачными САПР, решают данную проблему. Весь дистрибутив системы находится вне пользовательских устройств, что позволяет вносить изменения в систему без взаимодействия с пользователем.

Интеграция с другими системами. Взаимодействие всех систем автоматизации бизнес-процессов позволяет создать единое информационное пространство предприятия, что в свою очередь повышает эффективность разработки и выпуска продукции. Система, формирующая такое информационное пространство, называется системой управления жизненным циклом изделия (PLM). Она включает в себя систему управления ресурсами предприятия (ERP), систему управления взаимоотношениями с клиентами (CRM), систему управления цепями поставок (SCM), и область автоматизации проектирования [70]. В данной области выделяют несколько систем, три из которых в русской терминологии называются САПР: конструкторской подготовки (CAD), технологической подготовки (CAM), инженерных расчетов (CAE). Для объединения всех видов САПР была создана система управлением данными об изделии (PDM). Сегодня не все отрасли имеют возможность создания PLM [71]. Системы ERP, CRM, SCM универсальны и подходят для любого вида деятельности, проблемы их взаимодействия между собой уже решены. В области же САПР взаимодействие между системами все еще является проблемой. Эксперты в области внедрения универсальных и машиностроительных САПР называют интеграцию PDM систем «головной болью» [72]. Для обувных систем проблемой является обмен данными между

САПР конструкторской и технологической подготовки [73]. Веб-технологии, используемые в облачных САПР, не привязаны к каким-либо пользовательским системам или платформам, что позволяет сторонним разработчикам использовать общий интерфейс для взаимодействия с САПР.

Возможность применения «тяжеловесных» технологий. Перенос вычислений на серверную часть позволяет применить технологии, требовательные к вычислительным мощностям. К таким технологиям можно отнести техническое зрение, машинное обучение, нейросети, экспертные системы, средства имитационного моделирования и т.д.

Рассмотрим детально, что представляет собой облачная САПР обуви.

1.4 Разработка концепции облачной САПР обуви

Предоставление облачных услуг осуществляется по нескольким моделям [74] (рис. 1.7).

Наиболее распространенная из них – Software as a Service (SaaS), программное обеспечение как услуга. Это наиболее простая модель, в которой потребителю предоставляется доступ к программе по сети. По возможности, создатели таких программ разделяют ее на модули, чтобы потребитель мог оплачивать только те функции, которыми пользуется. Модель Platform as a Service (PaaS) – платформа как услуга, подразумевает предоставление потребителю доступ к программным системам, необходимых для работы программного обеспечения.

Например, системы хранения данных, платформы запуска программного обеспечения, а также различные средства разработки, тестирования и мониторинга систем.

Infrastructure as a Service (IaaS) – инфраструктура как услуга. Модель, при которой потребителю предоставляется виртуальный сервер заданной мощности. Установка каких-либо программных средств, включая операционную систему, остается полностью за потребителем. Чаще всего, на серверах размещается программное обеспечение, которое поставщик облачных технологий не может предоставить по модели PaaS.

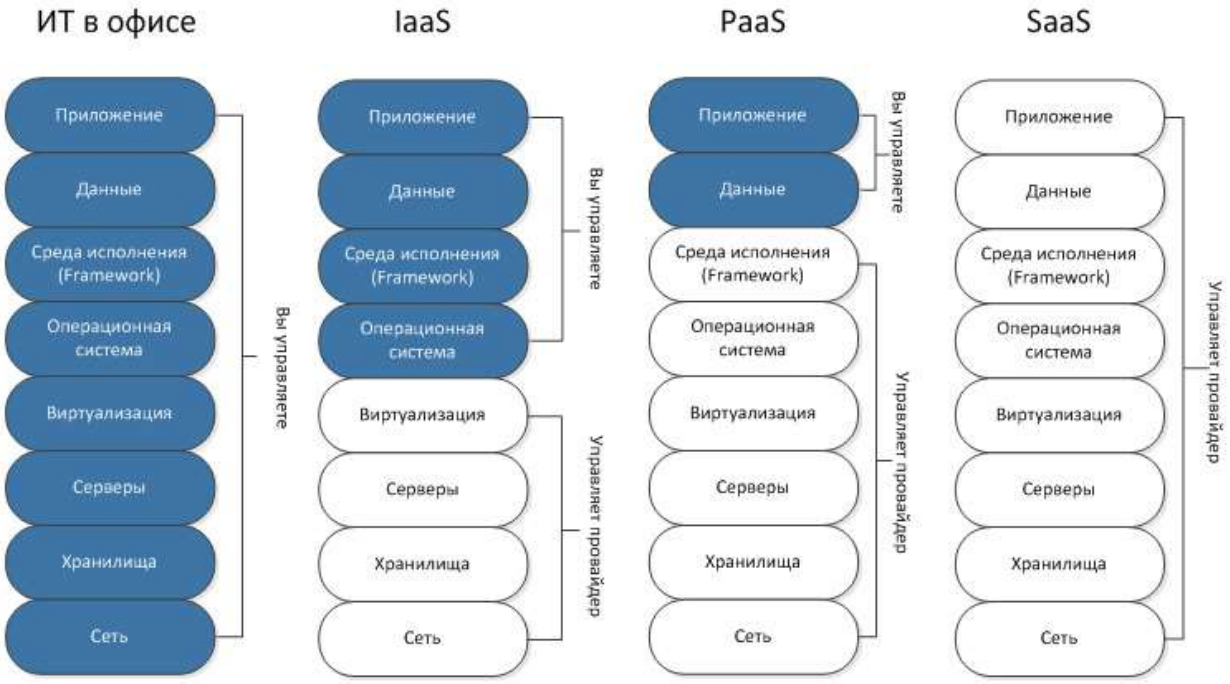


Рисунок 1.7 Модели обслуживания облачных технологий [74]

Участники всех организационных процессов разработки, внедрения и применения облачной САПР могут быть разделены на 3 роли по критерию владения: поставщик, разработчик и пользователь.

Поставщик облачных технологий — это владелец физических серверов, предоставляющий доступ к их вычислительным мощностям.

Разработчик — это владелец системы автоматизированного проектирования, использующий вычислительные мощности поставщика облачных технологий.

Пользователь — это владелец данных, произведенных в результате работы системы автоматизированного проектирования.

При этом роли не исключают друг друга: один и тот же участник может выступать в нескольких ролях. В соответствии с таким представлением можно сформировать модели развертывания системы [75] (рис.1.8).

Публичное облако (Public Cloud) - наиболее распространенная модель, при которой поставщик является отдельным лицом, не включенным в процессы разработки или применения облачной САПР. Разработчики системы приобретают у поставщика вычислительные мощности по модели PaaS или IaaS, в зависимости от требований к системе, и предоставляют конечному пользователю доступ по модели SaaS. Поставщик берет на себя ответственность за физическую сохранность данных, безопасность и отказоустойчивость предоставляемой инфраструктуры. Разработчик также несет ответственность за сохранность данных и работоспособность САПР.

Частное облако (Private Cloud) - модель, при которой роли поставщика и пользователя приходятся на одного участника - предприятие. Разработчики размещают САПР на оборудовании предприятия, и осуществляют контроль за системой удаленно. Такой подход позволяет предприятию не только полностью контролировать систему, но и задействовать свое собственное оборудование, накопленное за время работы предприятия.

Гибридное облако (Hybrid Cloud) – это комбинация публичного и частного облака, при которой в роли поставщика выступает как отдельное лицо, так и предприятие, использующее САПР. Часть инфраструктуры системы размещается на оборудовании поставщика облачных услуг, а другая часть – на оборудовании предприятия, например база данных. Такой подход позволяет

предприятию, не имеющих избыточных вычислительных мощностей, осуществлять частичный контроль за данными системы.

Рисунок 1.8 Модели развертывания облачных технологий [75]

Технически, облачная САПР делится на клиентскую и серверную части. Клиентская часть, выполняемая на компьютере пользователя, представляет собой веб-приложение. Основное преимущество веб-приложения заключается в том, что его запуск производится через веб-браузеры, которые сегодня предустановлены на всех распространенных операционных системах [76]. Это позволяет решить проблемы кроссплатформенности, развертывания, удаленной работы, защиты дистрибутива. Визуализация графики представляет собой ресурсоемкую задачу, которую раньше трудно было решить без специального программного и аппаратного обеспечения. Большинство современных

подходов в значительной степени полагались на расширения браузера, такие как Java 3D, JavaFX, O3D, Adobe Flash, ActiveX, Unity и другие [77]. Однако применение расширений браузера имеют недостатки [78]. Во-первых, расширения требуют отдельной установки, и пользователи не всегда могут осуществить ее самостоятельно. Также, многие распространенные платформы, например браузеры для мобильных устройств или планшетов не имеют поддержки расширений. Во-вторых, описанные расширения используют свою собственную модель визуализации, доступ к которой нельзя получить из кода веб-страницы. В настоящее время для разработки веб-сайтов, используется стандарт HTML5. Он отличается от предыдущих версий тем, что не требует каких-либо проприетарных расширений для вывода сложной графики, поддерживая при этом широкий спектр устройств [79]. В частности, элемент Canvas предоставляет возможность выполнения кода шейдера прямо в браузере, благодаря технологии WebGL. Это технология использует возможности OpenGL для веб-сайтов, обеспечивая аппаратное ускорение вывода графики [80]. Для рендеринга интерактивной графики используется встроенный в браузер яызк JavaScript, без необходимости установки дополнительных программных расширений. Множество исследователей изучали потенциал WebGL и HTML в различных областях, включая САПР [81]. Облачные САПР OnShape и позже Fusion360 были построены с помощью этой технологии [82]. Применение WebGL не отличается от OpenGL, что позволяет использовать существующие методы разработки пользовательского интерфейса САПР [83,84].

Реализация серверной части имеет ряд сложностей ввиду того, что компоненты облачной системы географически разрознены и независимы друг от друга [85]. Игнорирование этого обстоятельства в корне меняет способ разработки программных систем. В 90-х годах прошлого века сотрудник

компании Sun Microsystems Дейч П. сформулировал 8 факторов, которые необходимо учесть при разработке распределенной системы [86]:

1. сеть может стать причиной ошибок;
2. существуют задержка передачи данных по сети;
3. полоса пропускания ограничена;
4. сетевое взаимодействие требует дополнительных мер безопасности;
5. расположение вычислительных узлов в сети может быть изменено;
6. систему обслуживают множество администраторов;
7. транспортная инфраструктура имеет издержки;
8. сеть неоднородна.

Помимо этого, проектирование распределенных систем является трудной задачей по перечисленным ниже причинам [87].

- разработчикам приходится комбинировать множество вариантов возможных ошибок, а не их условий, так как зачастую появление одной ошибки не зависит от появления другой ошибки;
- запросы, требующие ответа, могут никогда его не получить;
- на любой сетевой запрос может быть получен "неизвестный" ответ, когда запрос получен, но не обработан по неизвестным причинам.
- ошибки могут возникнуть на уровне физических компьютеров, а не только на программном уровне;

Облачные системы сложны в тестировании из-за необходимости проведения дополнительных видов тестирований перед выпуском каждой новой версии системы [88].

Тестирование облачной безопасности позволяет убедиться в том, что данные предприятия хранятся и передаются безопасно. Тестирование проводится с использованием методов и инструментов, которые применяются

хакерами на практике. Так можно гарантировать безопасность облачных систем.

Тестирование масштабируемости и производительности позволяет определить скорость отклика, надежность, пропускную способность и масштабируемость системы при заданной рабочей нагрузке.

Тестирование облачной нагрузки и стресс-тестирование направлено на оценку работы системы при высоких уровнях нагрузки. Для любой системы важно работать даже в тяжелых условиях и сохранять стабильность. Стресс-тестирование производится путем создания пиковых нагрузок специальными программами-симуляторами.

Тестирование задержки оценивает время между запросами и ответами для всех действий приложения.

Тестирование аварийного восстановления выполняется для измерения времени восстановления системы после сбоя, а также оценки количества потерянных данных. Кроме того, существуют факторы, осложняющие процесс отладки в распределенной среде [89].

Неоднородность - как вычислительные, так и клиентские узлы распределенной системы могут включать устройства различных конфигураций - смартфоны, ноутбуки, различные серверы и т. д. Такое разнообразие аппаратного и программного обеспечения порождает множество проблем совместимости.

Конкурентность - одновременная работа нескольких узлов приводит к состояниям взаимоблокировок, которые сложно диагностировать. Кроме того, сеть имеет задержку, что усугубляет задачу отладки.

Системы защиты - облачные системы обладают множеством систем защит от различных бедствий. Сюда относятся не только системы защиты от

внешних угроз, но и системы защиты от сбоев. Все эти системы оставляют свой отпечаток в каждом процессе, что сильно усложняет отладку. Трудозатраты, приходящиеся на этапы разработки САПР, составляют основную часть ее стоимости. Непродуктивный процесс разработки системы может лишить преимущество низкой стоимости, перед обычной САПР [90]. В свою очередь, решить проблемы высокой сложности разработки можно на этапе проектирования системы, разработав оптимальную архитектуру системы [91]. Перенос вычислений с клиентского устройства на серверную часть, совместно с эффективной разработкой позволит не только адаптировать систему под быстроменяющиеся условия эксплуатации, но и открывает возможности для экспериментов с новыми технологиями. Так, применение средств технического зрения и векторизации позволит решить проблему оцифровки чертежа в условиях дистанционной работы.

Таким образом, вышеизложенное позволяет говорить о необходимости разработки концепции облачной системы автоматизированного проектирования обуви, которая составит теоретическую основу САПР и позволит обеспечить быструю и эффективную реализацию ее серверной части. Концепция должна определять, структуру, выполняемые функции и взаимосвязь базовых компонентов облачной САПР обуви. Детализация этих компонентов, разработка метода оцифровки чертежа с применением средств технического зрения, включая алгоритм применения данного метода в рамках облачной САПР обуви – позволят решить проблему применения системы в режиме дистанционной работы.

Так как архитектура системы определяет форму и процесс работы компонентов, то следующая глава посвящена разработке архитектуры облачной САПР обуви.

Выводы по первой главе

1. Рассмотрена проблема удаленной работы в системе автоматизированного проектирования обуви. Показано, что стоимость системы, ее привязка к аппаратному ключу и низкая эффективность существующих методов оцифровки являются препятствиями для применения САПР обуви в режиме удаленного доступа.
2. Рассмотрены облачные технологии, их свойства, преимущества и недостатки. Проанализирован текущий уровень развития САПР в области облачных технологий. Показано, что существующие облачные САПР имеют машиностроительное направление и не могут быть применены для проектирования обуви.
3. Показано, что преимуществами облачных САПР общего назначения перед десктопными САПР обуви являются кроссплатформенность, быстрое развертывание и обновление, отсутствие необходимости содержать аппаратный ключ защиты, простая интеграция с другими системами, меньшая стоимость, возможность применения технологий, требовательных к вычислительной мощности, что позволяет решить проблему удаленного применения САПР обуви.
4. Дано общее представление об облачной САПР обуви: основные участники, их связь с моделями обслуживания и развертывания системы. Выявлено, что неоптимальная структура серверной части облачной САПР может привести к затруднениям при разработке, сопровождении, тестировании, отладке и отрицательно скажется на стоимости системы.
5. Для обеспечения эффективного внедрения облачных технологий в САПР обуви и ее работы в режиме удаленного доступа предложена концепция, предполагающая разработку: модели, структуры, выполняемых функций и взаимосвязи компонентов облачной САПР обуви, а также метода оцифровки чертежа с применением современных средств технического зрения и векторизации.

2. РАЗРАБОТКА АРХИТЕКТУРЫ ОБЛАЧНОЙ САПР ОБУВИ В КОНТЕКСТЕ ПРЕДЛАГАЕМОЙ КОНЦЕПЦИИ

Архитектура программного обеспечения заключает в себе ряд важных решений об организации программной системы, среди которых выбор структурных элементов и их интерфейсов, составляющих и объединяющих систему в единое целое; поведение, обеспечиваемое совместной работой этих элементов; организацию этих структурных и поведенческих элементов в более крупные подсистемы [92]. Архитектура влияет на скорость разработки и качество системы [91]. Каждое из этих направлений характеризуется множеством критериев. Так, на скорость разработки влияют [93]:

- **сложность разработки -** каждая архитектура стремится структурировать и упорядочить элементы системы, что упрощает процесс внедрения новой функциональности в систему;
- **сложность сопровождения -** изменения функциональных возможностей системы могут быть инициированы не только в процессе разработки, но и в процессе эксплуатации. Данный критерий отражает насколько быстро можно изменить систему для адаптации к новым условиям;
- **сложность тестирования -** любое тестирование системы направлено на проверку ее качества. Соответственно, чем легче производится тестирование, тем качественнее будет система;
- **сложность отладки -** при разработке сложных систем появление неизбежно ошибок. Критерий отражает, насколько быстро можно обнаружить и устранить ошибки;
- **сложность восприятия -** критерий, влияющий на сложность разработки и сопровождения. Процесс разработки может затянуться на долгое время, за которое может измениться штат разработчиков. Архитектура должна быть

легко восприниматься, чтобы новые разработчики могли быстрее приступить к разработке;

- **сложность документирования -** процесс документирования необходим для сопровождения системы, а также интеграции с другими системами. Сложности в разработке документации, чаще всего, вызваны скрытыми связями между элементами системы. Архитектура должна структурировать элементы так, чтобы количество таких связей было минимальным.

Другое направление архитектуры - качество системы. Представим наиболее важные для облачных систем [94].

Развертываемость - критерий, определяющий как быстро система может быть размещена на платформе и подготовлена к запуску.

Масштабируемость - наиболее важный критерий для облачных технологий - способность работать в условиях репликации вычислительных узлов - горизонтального масштабирования. Создание реплики системы или ее части на другом вычислительном узле позволяет равномерно распределить нагрузку на всю систему. Однако реплики, обращающиеся к одним и тем же данным, могут конфликтовать между собой. Хорошо спроектированная архитектура должна обеспечить работу системы без таких конфликтов.

Отказоустойчивость - способность системы работать частично или полностью после сбоя. К этому же критерию можно отнести и способность системы самостоятельно восстанавливаться после сбоя.

Производительность - наиболее важный критерий для систем автоматизированного проектирования, характеризующий время отклика системы на действия пользователя.

Совместимость - способность системы к взаимодействию с другими системами.

Основными архитектурными элементами являются модули, компоненты и артефакты [95]. Ввиду того, что методы разработки и языки программирования изменяются очень быстро, определения этих понятий часто меняются. В данной работе нами приняты следующие определения:

- модуль – выделенная по тем или иным мотивам часть системы [96];
- компонент — это модуль, обладающий состоянием и четко определенным интерфейсом взаимодействия с другими компонентами [97];
- артефакт - физический объект на уровне платформы реализации, например, исполняемый файл или динамическая библиотека [96].

Необходимость разработки системы, как набора модулей обусловлено множеством причин, в числе которых [98]:

- **сокращение записи -** повторяющиеся или близкие по содержанию части программы выносятся в отдельный модуль. Исходный код программы сокращается, за счет применения механизмов обращения к модулю, вместо текста алгоритма. Соответственно, такие модули оформляются в виде подпрограмм, макроопределений и т. п;
- **сокращение времени компиляции -** если модуль размещен в отдельном артефакте, то суммарное время компиляции уменьшается, так как при внесении изменений в перекомпиляции нуждается только затронутый модуль;
- **наглядность -** для удобства редактирования исходного кода, программу разбивают на более мелкие части, которые выступают в качестве обозримых и потому более удобных объектов редактирования. В то время как при модуляризации программы наглядность редко оказывается на первом месте. При разработке же текстовой документации соображения удобочитаемости, напротив, обычно становятся решающими;

- **распределение работ –** несколько разработчиков могут распределить работу над системой, разбитой на модули. Для сокращения количества обсуждений, проводимых в ходе разработке, в модулях минимизируется количество взаимосвязей;
- **сегментирование** – объем всей программы может быть разделен на модули для учета ограниченного количества оперативной памяти. Так, в ходе работы, активные модули сменяют друг друга;
- **систематизация и упрощение разработки –** так как модули улучшают наглядность, разработчику проще абстрагироваться от построения системы и сосредоточиться на решении конкретной задачи. Такое разбиение положительно влияет на сопроводительную документацию, которая в результате будет повторять структуру программы, отражая связь между текстами документов и программ.

В процессе разработки, чаще всего, модули формируются по единому для всех принципу: модуль должен представлять собой функционально самостоятельную часть программы. Такая общность происхождения приводит к тому, что границы модулей, вычленяемых по различным мотивам, легко согласуются между собой, в результате чего формируется гармоничная структура программы. Удобство разработки достигается за счет локализации изменений программы в одном модуле. Самостоятельный модуль нагляден, поскольку его назначение легко сформулировать. При этом в ходе реализации модулей облегчается взаимодействие разработчиков, так как модули могут разрабатываться параллельно [99].

По этому принципу можно выделить следующие функциональные модули САПР обуви:

- **модуль управления пользователями –** отвечает за управление учетными записями пользователей и обеспечивает их доступ к системе при помощи

механизма авторизации. В качестве данных идентификации пользователя традиционно используется пара логин-пароль;

- **модуль управления проектами -** решает задачи создания и редактирования общих данных о моделях обуви;
- **модуль моделирования -** реализует возможность проектирования в определенном формате и методе. Система может содержать множество модулей такого типа, каждый из которых будет представлять свой собственный метод проектирования и взаимодействия с пользователем;
- **модуль градирования** – занимается градированием модели обуви на размерный ряд;
- **модуль раскроя -** позволяет получить карту раскроя деталей обуви;
- **модуль импорта -** реализует работы системы с внешними данными, путем их конвертации в формат САПР обуви;
- **модуль оцифровки -** частный случай модуля импорта, при котором внешние данные представляют собой растровое изображение;
- **модуль экспорта -** занимается выгрузкой проектных данных в определенном формате;

Для проектирования архитектуры облачной САПР обуви необходимо определить методы взаимодействия между перечисленными модулями, пользователем и источниками данных. Отправной точкой в решении этой задачи является анализ существующих архитектурных шаблонов [96]. В следующих разделах диссертации представим более детальный анализ архитектурных паттернов, применительно к САПР обуви.

2.1 Применение монолитного архитектурного шаблона при разработке САПР обуви

Наиболее распространенная архитектура - монолитная, предполагающая размещение всех модулей программы в одном исполняемом файле [96]. Такой подход является традиционным для разработки программ [100]. Предлагаемая нами структура облачной САПР обуви в монолитном шаблоне (рис.2.1) демонстрирует хаотичную связь модулей системы, сравнимой с отсутствием архитектуры. Однако и такой подход применяется на практике, зачастую для создания прототипа системы, так как в ходе разработки выясняются неявные связи между модулями [101].

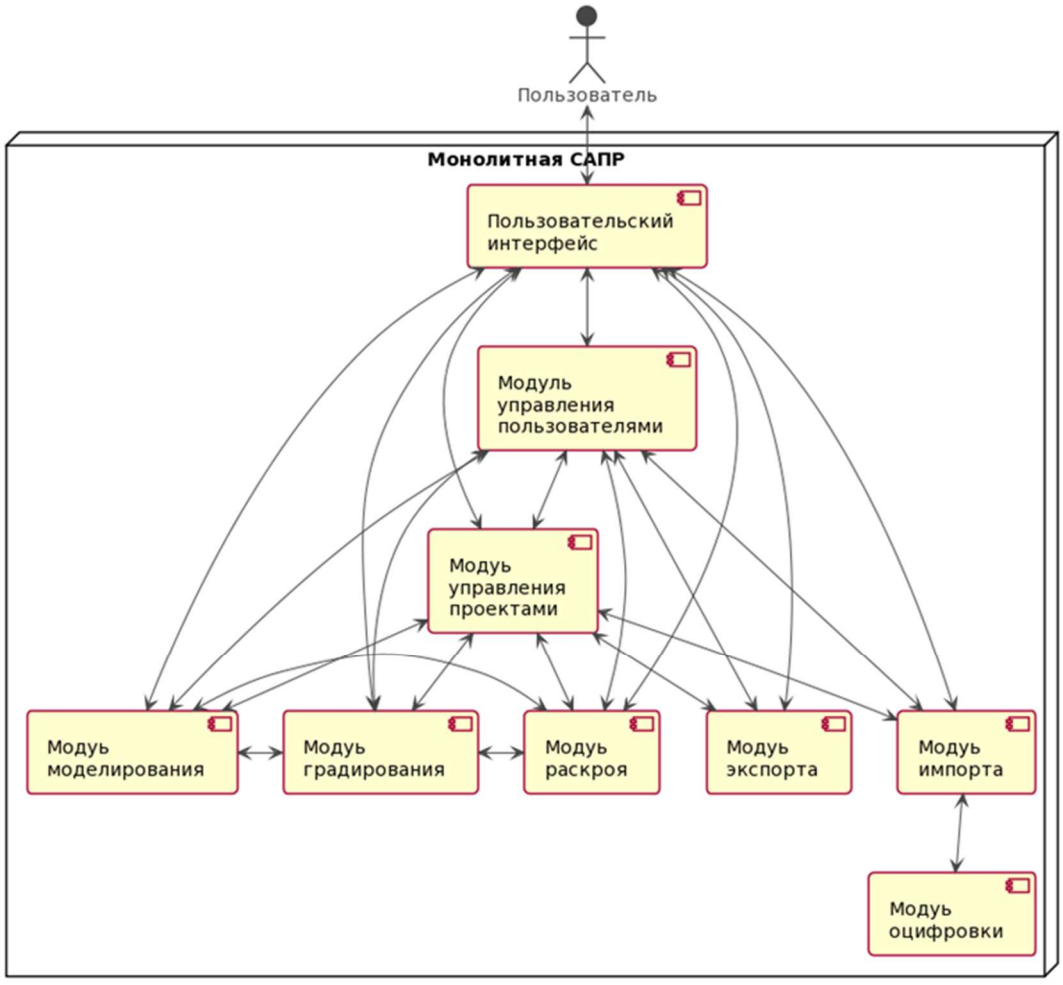


Рисунок 2.1 Монолитная архитектура САПР обуви

Для упрощения процесса разработки, известны способы упорядочения монолитной архитектуры по уровням. Один из первых и наиболее распространенных способов разделения монолита - трехуровневая архитектура, располагающая модули на одном и трех уровнях [102]:

- **уровень представления -** обеспечивает взаимодействие с пользователем приложения — это пользовательский интерфейс и уровень обмена данными. Основные задачи модулей на этом уровне состоят в отображении информации и получении информации от пользователя;
- **уровень приложений -** также известный как логический, является центральным звеном приложения. На этом уровне модули обрабатывают информацию, полученную из модулей других уровней, реализуя, таким образом, функциональные возможности всей системы;
- **уровень данных -** предназначен для хранения и управления информацией, обработанной приложением.

В трехуровневой архитектуре обмен данными осуществляется только через уровень приложения. Уровень представления и уровень данных не могут взаимодействовать друг с другом напрямую. Такой подход вводит правила, по которым каждый новый модуль должен быть отнесен к определенному уровню, что обуславливает способ его взаимодействия с другими модулями. Это упрощает разработку и вносит большую наглядность в архитектуру системы [103]. Предлагаемая нами структура монолитной трехуровневой облачной САПР обуви представлена на рисунке 2.2.

Более современный способ разделения монолитной системы предложил Мартин Р в 2012 году [104], назвав его «чистой архитектурой» (рис. 2.3). Окружности на диаграмме представляют собой различные уровни программного обеспечения, которые взаимодействуют друг с другом по

правилу зависимостей. В исходном коде зависимости могут быть направлены только во внутрь.

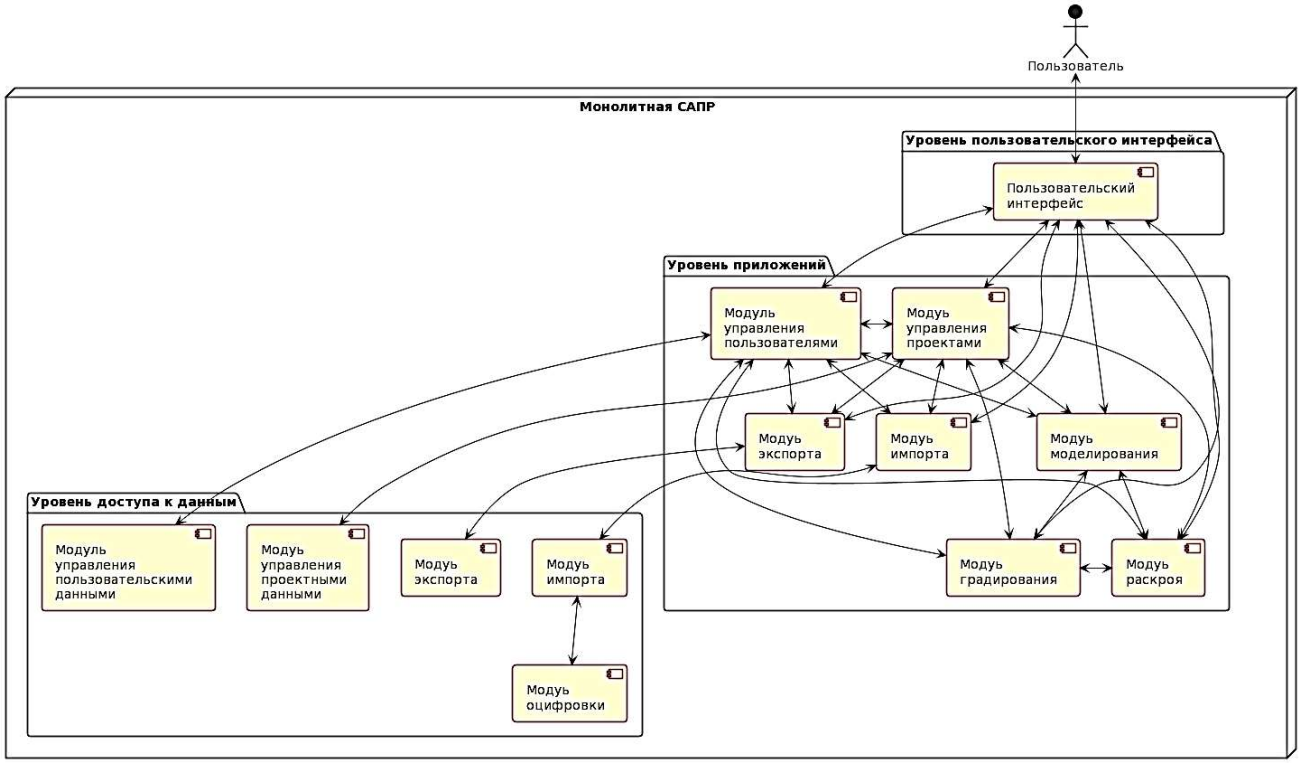


Рисунок 2.2 Монолитная трехуровневая архитектура САПР обуви

Модули из внутреннего круга не могут что-либо знать о модулях внешнего круга. Более того, структуры данных, используемых во внешнем круге, не должны быть использованы во внутреннем [104].

Программа рассматривается на 4-х уровнях:

- **сущности –** уровень содержит структуры данных, описывающие предметную область, а также различные модули, выполняющие обработку этих структур данных;
- **сценарии -** на этом уровне реализуются функциональные возможности системы, используя структуры данных и модули из уровня сущностей. Уровень сценариев — это место, где модули формируют поведение программы;

- **интерфейс-адаптеры –** это слой, который преобразует данные в формат, необходимый для дальнейшего применения, в зависимости от направления данных. Например, данные, проходящие в данный слой извне, будут преобразованы в формат, необходимый для работы сценариев. После обработки из уровня сценариев поступят данные в виде ответа, которые также будут преобразованы в нужный формат;
- **фреймворки и драйверы -** наружный слой обычно состоит из внешних источников данных, таких как пользовательский интерфейс, база данных и т.д. Как правило, в этом слое не пишется много кода, за исключением общения с внутренними уровнями.

Применение такого разделения, на сегодняшний день, является лучшим способом структурирования монолитной архитектуры [105]. Предлагаемая нами структура облачной САПР обуви, построенной по принципам «чистой» архитектуры, изображена на рисунке 2.4.

Сложность разработки монолитной архитектуры растет пропорционально росту количества модулей. Внедрение новой функциональности зачастую требует пересмотра связей между модулями [106]. При этом сложность тестирования и отладки на протяжении всего процесса разработки остается низкой. Кодовая база монолитной системы едина, что позволяет использовать все возможности интегрированных сред разработки для анализа и построения системы.

Все модули размещаются в одном пространстве памяти, что упрощает процесс отладки и тестирования системы [101].

Сложность сопровождения напротив - высока. С ростом системы среда разработки может не справляться со всей кодовой базой [106]. Модули системы могут обзавестись сильными связями между собой, что потребует внесения

изменений во все модули по цепочке зависимостей. При этом, скрытые и неявные связи между модулями могут стать причиной ошибок при внесении каких-либо изменений [107].

Рисунок 2.3 «Чистая архитектура», предложенная Робертом Мартином [104]

Монолитная архитектура в структурированных многоуровневых вариантах легко воспринимается, что позволяет новым разработчикам быстро приступать к работе. Однако с ростом количества модулей системы, структурированность играет меньшую роль, чем документация. В свою очередь, процесс документирования осложнен из-за взаимосвязей между модулями [108]. Большое количество модулей в монолитной архитектуре усложняет процесс развертываемости и масштабируемости. Так как во время работы растет объем памяти, занимаемый как на физическом носителе, так и в

оперативной памяти. Масштабирование монолитной системы приумножает количество необходимых вычислительных ресурсов, хотя нагрузку может испытывать не вся система, а только некоторые модули [101].

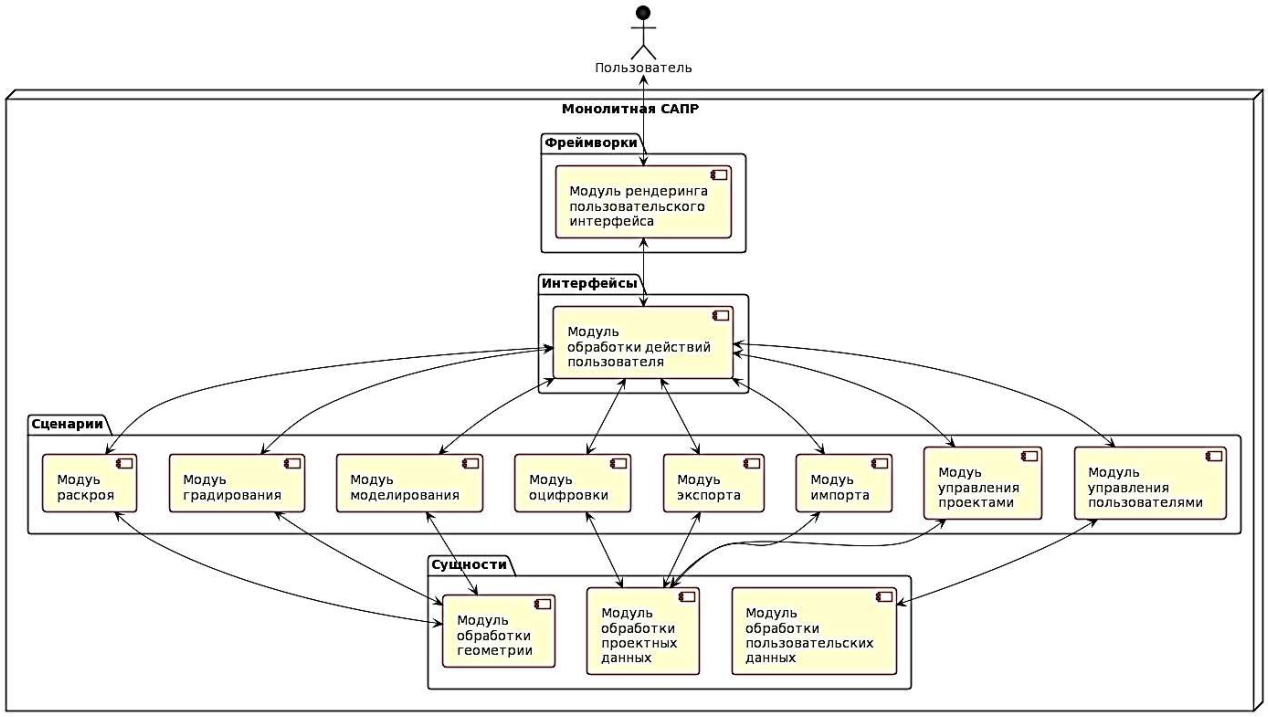


Рисунок 2.4 Монолитная архитектура САПР обуви, структурированная по принципам «чистой архитектуры»

Отказоустойчивость может быть заложена в системе, но только по отношению к программным сбоям. Если сбой произойдет на уровне выполняющей платформы, операционной системы или вычислительного устройства, то работа монолитной системы также будет остановлена [109].

Совместимость на уровне архитектуры отсутствует, механизм взаимодействия с другими системами необходимо реализовывать в процессе разработки [107].

Лучшее, что может предложить монолитная архитектура - производительность. Ввиду того, что все модули располагаются в одном

пространстве памяти, взаимодействие между ними происходит самым быстрым способом из возможных [110].

Из-за проблем с масштабированием и замедлением процесса разработки по мере увеличения количества модулей, монолитная архитектура не пригодна для построения облачной системы автоматизированного проектирования обуви.

2.2 Структуризация САПР обуви при помощи микроядерного архитектурного шаблона

Как было описано раннее, одна из причин разбиения системы на модули - сегментация, предполагающая последовательную загрузку модулей в оперативную память, по мере их использования. Возможность такой загрузки со временем стали использовать для создания расширяемых программ.

Для реализации механизма подключения модулей необходимо разместить каждый модуль в отдельном артефакте, а также реализовать в модулях единый интерфейс взаимодействия с системой. При инициализации расширений, система сканирует артефакты на предмет реализации интерфейса, после чего загружает модуль в память. Чаще всего, такой подход применялся для того, чтобы сторонние разработчики могли расширить функциональность системы на свое усмотрение [111].

Реализация всех модулей системы, как динамически подключаемых, приведет к формированию микроядерной архитектуры. Она состоит из ядра и подключаемых модулей, между которыми разделена логика приложения. Модули автономны и независимы, то есть имеют форму компонентов. Они выполняют обработку данных, реализуя задачи предметной области. В свою очередь, ядро содержит только минимальный набор функций, необходимых для обеспечения работоспособности системы. Как правило, это инфраструктурная

логика, без какой-либо сложной обработки данных [112]. Например, ядро, имея информацию о том, какие подключаемые модули доступны, должно загрузить их. Один из распространенных способов реализации этого - создание реестра подключаемых модулей. Такой реестр содержит информацию о каждом модуле, включая такие вещи, как его имя, интерфейс взаимодействия, уровне безопасности [113]. Предлагаемая нами структура микроядерной облачной САПР обуви изображена на рисунке 2.5.

Микроядерная архитектура обладает высокой сложностью разработки на начальном этапе, так как создание механизма подключаемых модулей и определение минимально необходимого набора функций ядра представляет собой непростую задачу. Однако дальнейшая разработка–облегчается за счет создания модулей по одному и тому же принципу [112].

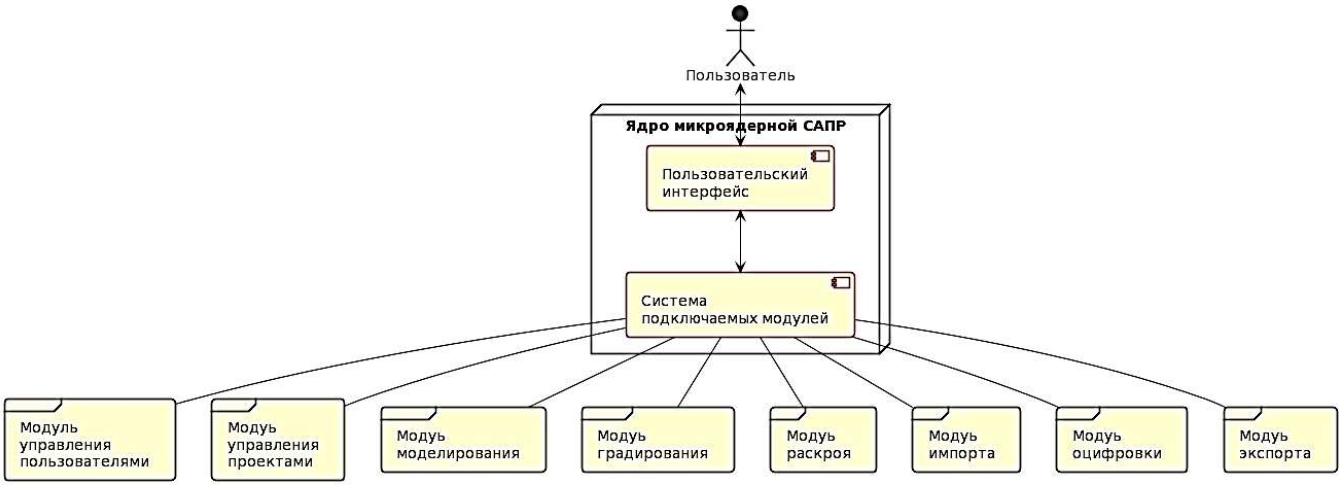


Рисунок 2.5 Микроядерная архитектура САПР обуви

Процесс сопровождения зависит от взаимодействия модулей между собой. Если модули изолированы и самодостаточны, то сопровождение имеет низкую сложность, так как внесение изменений в один модуль не повлечет за

собой изменения в другом. Вместе с тем, необходимо отметить, что внесение изменений в ядро может потребовать перестроения всех модулей [114].

Модули и ядро монолитны, что упрощает их тестирование по отдельности. Однако, интеграционное тестирование требует создания специального тестировочного ядра [115].

Отладка микроядерной системы имеет среднюю сложность, так как, помимо поиска собственно ошибки, необходимо выполнить поиск модуля, в котором она произошла. Проблемы отладки решаются более строгими требованиями к построению модулей. Например, вводя обязательную систему перехвата ошибок и отчета о них ядру системы [116].

Порог вхождения в микроядерную архитектуру для новых разработчиков очень низкий. Так, для начала разработки достаточно изучить интерфейс взаимодействия с ядром [112].

Сложность документирования микроядерной системы зависит от взаимосвязей между модулями. Если модули самодостаточны, то процесс документирования ограничится описанием алгоритмов модулей [117].

Развертываемость микроядерных систем значительно лучше монолитной, так как все части системы развертываются независимо друг от друга. Существует возможность развернуть минимально функциональную систему, и затем подгружать новые модули по мере их разработки [118].

Масштабируемость сильно зависит от связи между модулями. Если модули самодостаточны и не взаимодействуют между собой через ядро, то есть возможность масштабировать систему в виде ядра и ограниченного количества модулей, испытывающих наибольшую нагрузку [112].

Микроядерные системы могут обладать отказоустойчивостью. Для этого необходимо реализовать механизм перехвата ядром ошибок модулей. Так, ядро

может отключить проблемный модуль, освободить память и загрузить его заново [119].

Производительность данной архитектуры немного хуже, чем монолитной, так как взаимодействие пользователь-модуль или модуль-модуль осуществляется через посредника - ядро. Однако возможная задержка отклика минимальна и зависит только от объемов передаваемых данных [120].

Совместимость не заложена архитектурой, однако ее реализация гораздо проще, чем у монолитных систем. Для взаимодействия напрямую с ядром достаточно создать отдельную точку входа в систему [112].

Таким образом, в области облачных систем микроядерная архитектура имеет преимущества по сравнению с монолитной, в частности: возможность гибкого масштабирования и отказоустойчивость. Процесс разработки может быть замедлен на этапе проектирования окончательной структуры архитектуры, однако собственно разработка и сопровождение функциональности системы по скорости не уступают монолитной архитектуре. Все эти преимущества возможны только при условии самодостаточности модулей. Большую систему, каковой является облачная САПР обуви, сложно построить без взаимодействия между модулями, поэтому данная архитектура не подходит для реализации задач в рамках диссертации.

2.3 Распределение компонентов САПР обуви по вычислительным узлам, используя сервис-ориентированный архитектурный шаблон

Распространение информационных технологий в корпоративном секторе стало причиной возникновения потребности в стандартном способе взаимодействия приложений, которые использовали разные технологии, и выполняются на разных вычислительных устройствах. В таких условиях

работали организации, разделенные на отделы и департаменты, каждый из которых обладал собственными системами обработки информации. Отделы передавали друг другу информацию в формате, определенном принимающим отделом. По той же схеме была сформирована сервис-ориентированная архитектура информационных систем [121]. Технически, она похожа на микроядерную тем, что модули располагаются в отдельном артефакте. Однако, такой артефакт называется «сервис», и имеет форму исполняемого файла, а не динамической библиотеки. Другими словами, сервис-ориентированная система реализована в виде некоторого количества программ, которые доступны извне пользователям и другим сервисам через соответствующие коммуникационные механизмы. В качестве таких механизмов используются сетевые протоколы [122].

Основное преимущество применение сервисов в качестве основных структурных элементов системы – масштабирование [123]. Каждый сервис может быть расположен на отдельном вычислительном узле. Сервисы, испытывающие высокие нагрузки, могут быть горизонтально масштабированы - запущены на новых вычислительных узлах для распределения нагрузки.

Коммуникация между сервисами и пользователями играет ключевую роль в разработке такой системы. Рассмотрим подробно проблемы коммуникации и методы их решения.

Существует два стиля взаимодействия между клиентом и сервисом [124]:

- синхронное – в формате «запрос-ответ». Клиент отправляет сервису запрос и ждет ответа. Этот стиль взаимодействия обычно приводит к созданию жесткой связи между сервисами.
- асинхронное – подразумевает, что клиент не блокируется, а ответ, если таковой придет, может быть отправлен не сразу.

Представим их более подробнее.

Для синхронного взаимодействия используется механизм удаленного вызова процедур (remote procedure call, RPC). Принцип работы RPC показан на рисунке 2.6.

Вызывающий сервис обращается к интерфейсу взаимодействия, реализованному RPC-клиентом. RPC-клиент выполняет запрос к сервису. Этот запрос обрабатывается RPC-сервер, который запускает необходимые процессы в сервисе. Затем ответ передается обратно RPC-клиенту, который возвращает результат вызывающему сервису [125]. Клиент и сервер RPC работают по определенному коммуникационному протоколу. На сегодняшний день наиболее распространен протокол REST. Филдинг Р., создатель протокола, дает следующее определение этой технологии [126]: REST предоставляет набор архитектурных ограничений, которые, если их применять как единое целое, делают акцент на масштабируемости взаимодействия между компонентами, обобщенности интерфейсов, независимом развертывании компонентов и промежуточных компонентах, чтобы снизить латентность взаимодействия, обеспечить безопасность и инкапсулировать устаревшие системы.

Рисунок 2.6 Схема работы механизма удаленного вызова процедур (RPC)

Ресурс является ключевой концепцией в REST. Он представляет определенный вид объектов предметной области. Для работы с ресурсами REST использует HTTP-команды, которые указываются с помощью URL. GET-запрос возвращает представление ресурса. POST-запрос создает новый ресурс, а PUT-запрос обновляет существующий [127].

При проектировании REST часто возникает проблема обращения к нескольким родственным объектам за один запрос. Если строго следовать стандарту REST, то для этого потребуется выполнить несколько запросов, что сказывается на времени отклика [128].

Еще одна распространенная проблема проектирования REST связана с ограниченным количеством возможных видов HTTP – запросов. Количество действий, которые можно выполнить с ресурсом, может быть значительно больше, чем запросов [129]. Особенно, эта проблема касается САПР обуви. Также есть проблема с обновлением части ресурса, вызванная его слишком большим определением [130].

Эти проблемы привели к появлению альтернатив, таких как gRPC. Данный протокол представляет собой двоичное сообщение, описываемое–с помощью языка IDL на основе Protocol Buffers – многоязычного механизма сериализации структурированных данных от компании Google. Формат является маркируемым. Каждое поле сообщения нумеруется и содержит код типа. Получатель сообщения может извлечь нужные ему поля, и проигнорировать остальные, которые не распознает. В итоге gRPC позволяет развивать интерфейс взаимодействия с сохранением обратной совместимости [131]. Компилятор Protocol Buffer генерирует код для сериализации и десериализации данных на различных языках программирования: Java, C#, JavaScript, Go [132]. Протокол gRPC — это полноценная альтернатива REST,

хотя оба они представляют собой синхронные коммуникационные механизмы и, как следствие, страдают из-за проблем с частичным отказом [133]. Рассмотрим подробнее этот недостаток.

Каждый раз, когда сервис в системе делает синхронный запрос к другому сервису, возникает риск частичного отказа. Поскольку сервис является отдельным процессом, он может не ответить вовремя на запрос клиента или оказаться недоступным из-за сбоя. Кроме того, сервис может быть перегруженным и отвечать на запросы чрезвычайно медленно. Поскольку клиент блокируется в ожидании ответа, существует опасность того, что его собственные клиенты тоже окажутся заблокированными и так по цепочке откажет вся система.

Шаблон «предохранитель» (circuit breaker) предотвращает попытки приложения выполнить операцию, которая скорее всего завершится неудачно, что позволяет продолжить работу, дальше не тратя важные ресурсы, пока известно, что проблема не устранена. Сервис должен быстро принять сбой операции и обработать его [134]. «Предохранитель» имеет 3 состояния:

- **закрытое -** сервис посылает запросы. RPC-клиент ведет подсчет числа недавних сбоев. Если число недавних сбоев превышает заданный порог в течение заданного периода времени, RPC-клиент переводится в открытое состояние. На этом этапе запускается таймер времени ожидания, и по истечении времени RPC-клиент переводится в полуоткрытое состояние. Цель «предохранителя» – дать системе время на исправление ошибки, которая вызвала сбой, прежде чем разрешить приложению попытаться выполнить операцию еще раз.

- **открытое** - любой запрос к сервису, который испытывает проблемы, немедленно завершает с ошибкой и исключение возвращается в приложение.
- **полуоткрытый** - RPC-клиент разрешает отправлять ограниченное число запросов к сервису, который испытывает проблемы. Если эти запросы выполняются успешно, предполагается, что ошибка устранена, и «предохранитель» переходит в закрытое состояние, счетчик сбоев сбрасывается. Если какой-либо запрос завершается со сбоем, то предполагается, что неисправность все еще присутствует. «Предохранитель» возвращается в открытое состояние и перезапускает таймер времени ожидания.

Для выполнения запроса сервису необходимо знать местоположение экземпляра сервиса в сети (IP-адрес и порт). В облачных системах сетевой адрес экземплярам сервисов назначается динамически. Более того, набор этих экземпляров постоянно меняется из-за автоматического масштабирования, отказов и обновлений. В связи с этим необходимо использовать **механизм «обнаружение сервисов»** [135].

В общем случае **механизм обнаружения сервисов** аналогичен механизму поиска модулей в микроядерной архитектуре. Ключевым компонентом выступает реестр сервисов – база данных с информацией о том, где находятся экземпляры сервисов. Адрес данного реестра статичен и не меняется. При запуске сервис регистрирует себя в реестре. При запуске и остановке сервисов, механизм обнаружения обновляет реестр. После чего реестр периодически сообщает всем сервисам о внутренней структуре системы. Помимо этого, реестр занимается проверкой работоспособности сервисов (healthcheck). Отправляя сообщение, реестр ждет ответа, если он не поступит в

течение определенного времени - сервис будет удален из реестра и подаст сигнал платформе, что с сервисом возможны проблемы [135].

Если в системе применяется горизонтальное масштабирование, то необходим компонент балансировки нагрузки. Его задача инкапсулировать группы одинаковых сервисов, принимая все поступающие к сервисам запросы и затем распределять запросы по ним [136].

Балансировка на сетевом уровне предполагает использование IP-адреса сервиса или отправителя. К примеру, такая балансировка может осуществляться с применением системы доменных имен: на одно доменное имя выделяется несколько IP-адресов сервисов. В свою очередь, анализ адреса отправителя позволяет балансировать нагрузку по территориальному признаку.

Балансировка на транспортном уровне - самая простая, ввиду большого количества информации на этом уровне. Балансировщику нагрузки достаточно знать сетевые адреса для работы.

Балансировка на прикладном уровне - работает по аналогии с транспортным уровнем, но при этом имеется доступ к структуре и содержимому запроса. Анализ данных в запросе позволяет балансировать нагрузку в зависимости от характера запрашиваемых данных.

Из перечисленных вариантов, наиболее предпочтительно использовать балансировку нагрузки на прикладном уровне [137,138]. Проблема, которую может решить только такой подход – точное определение загруженности сервиса. Балансировщики на транспортном и сетевом уровнях перестанут перенаправлять запросы на сервис, который перестал отвечать. Однако это очень редкий случай. Веб-серверы чаще всего работают в отдельном потоке от алгоритмов обработки данных, что позволяет им отправить ответ независимо от работы этих алгоритмов. Даже в случае критического сбоя, веб-сервер может

отправить пользователю сообщение об ошибке. Балансировщик нагрузки на прикладном уровне сможет проанализировать этот ответ и перейти к принятию решения о дальнейшем перенаправлении запросов на загруженный сервис.

В сервис-ориентированной архитектуре взаимодействие путем асинхронного обмена сообщениями осуществляется через брокер сообщений [139].

Брокер — это промежуточное звено, через которое проходят все сообщения. Отправитель передает сообщение брокеру, а тот доставляет его получателю. Работа брокера сообщений основана на модели «издатель-подписчик». Сервисы регистрируют в брокере типы сообщений, который они могут отправлять, и, которые они хотят получать [140].

Используя эту модель, можно реализовать одностороннее взаимодействие, называемое оповещением. Сервис отправляет сообщению множеству других сервисов, не ожидая от них ответа.

Также реализуется двустороннее взаимодействие – асинхронный запрос с ответом. Взаимодействующие сервисы подписаны на сообщения друг друга. При отправке запроса в заголовке сообщения указывается уникальный идентификатор. Формируя ответ, сервис в заголовке сообщения указывает тот же идентификатор, чтобы сопоставить ответ с запросом. Сервис-отправитель может заблокироваться пока не получит ответ, чтобы имитировать синхронное взаимодействие. Однако на практике обработка ответов происходит асинхронно, особенно, если ответ могут прислать несколько сервисов одновременно. Предлагаемая нами структура сервис-ориентированной САПР обуви, использующей брокер сообщений, представлена на рисунке 2.7

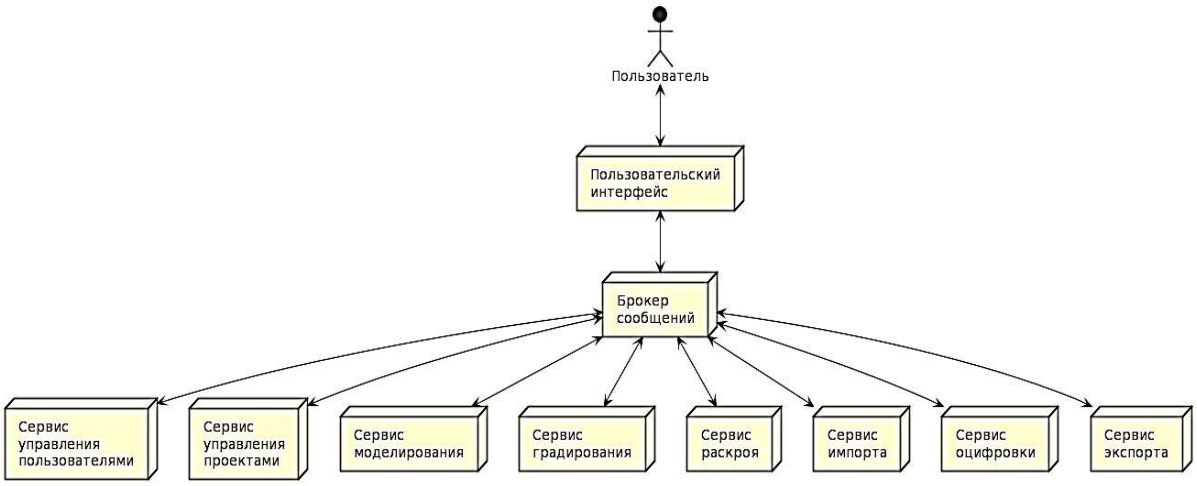


Рисунок 2.7 Сервис-ориентированная САПР обуви с применением брокера сообщений

Применение брокера сообщений имеет такие достоинства как [141]:

- **слабая связанность -** для выполнения запроса отправителю нужно лишь послать сообщение в брокер. Отправителю сообщения ничего не известно о внутренней структуре системы, и ему не нужно использовать механизм обнаружения, чтобы определить местонахождение сервисов.
- **буферизация сообщений -** брокер буферизирует сообщения до тех пор, пока их не смогут обработать. В протоколах с синхронным взаимодействием и отправитель, и получатель должны быть доступны на протяжении всего времени обмена данными. Брокер накапливает сообщения в очереди, пока получатель не будет готов их принять.
- **явное межсервисное взаимодействие -** механизмы, основанные на RPC, пытаются сделать так, чтобы обращение к удаленному сервису выглядело как вызов локальной функции. Однако сетевое взаимодействие и вероятность частичных отказов делают эти два вида взаимодействия очень разными. Обмен сообщениями через брокер делает различия явными,

чтобы у разработчиков не возникало ложное чувство безопасности. При этом у брокера сообщений есть и некоторые недостатки [141]:

- **производительность -** задержка передачи данных через посредника скажутся на времени отклика системы.
- **потенциальная единая точка отказа -** крайне важно, чтобы брокер сообщений был отказоустойчивым, иначе может пострадать надежность системы.
- **дополнительная сложность в администрировании -** механизм обмена сообщениями — это еще один системный компонент, который нужно устанавливать, конфигурировать и администрировать.

Разделение функциональности системы на сервисы усложняет выполнение транзакций — группы последовательных операций, которые представляют собой логическую единицу работы в базе данных. В сложных информационных системах транзакция состоит из действий, последовательно выполняемых различными модулями. Все транзакции должны следовать ACID - требованиям [142]:

- **атомарность (Atomicity) –** гарантия того, что транзакция выполнится полностью или не выполнится совсем;
- **согласованность (Consistency) –** гарантия сохранения только допустимых результатов, в случае нормального завершения транзакции;
- **изолированность (Isolation) –** выполнение каких-либо операций вне транзакции не должны влиять на результат ее выполнения;
- **стойкость (Durability) –** подтверждение о выполнении транзакции гарантируют, что какой-либо сбой не отменит внесенные изменения.

Для обеспечения согласованности данных между несколькими сервисами, базами данных или брокерами сообщений необходимо реализовать шаблон проектирование «сага» (saga) [143]. Идея данного шаблона, заключается в

разбиении распределенной транзакции на цепочку локальных транзакций. При обновлении базы данных каждая локальная транзакция публикует сообщение, которое провоцирует выполнение следующей транзакции. Если локальная транзакция по каким-либо причинам не была выполнена, то запускается цепочка компенсирующих локальных транзакций, которые откатывают произошедшие изменения в предыдущих транзакциях. Существует два метода координирования выполнения локальных транзакций в «сагах» [144].

В **оркеструемой** «саге» существует объект оркестратор, который поочередно запускает операции на сервисах и следит за ходом выполнения всей транзакции. В случае сбоя, оркестратор производит откат транзакции. Зачастую оркестратор реализуется как отдельный сервис для обеспечения высокой отказоустойчивости;

В **хореографической** «саге» сервисы выполняют локальную транзакцию и сообщает о результате следующему сервису-участнику распределенной транзакции. В случае сбоя, сервис отправляет выполняет откат и сообщает об этом предыдущему участнику транзакции. Таки образом формируется цепочка выполняемых операций транзакции.

В системе при выполнении локальных транзакций могут возникать два типа ошибок [143].

Ошибки обработки данных возникают, когда локальную транзакцию невозможно выполнить из-за несоблюдения правил предметной области. Подобные ошибки должны сразу приводить к запуску цепочки компенсирующих транзакций.

Технические ошибки, связаны с тем, что все взаимодействие между сервисами осуществляется по сети, где могут возникать временные проблемы с доступностью. Эти ошибки неизбежно будут возникать в распределенных

системах. Необходимо реализовать транзакции таким образом, чтобы их можно было повторно выполнить и получить тот же результат. Тогда при возникновении временных технических ошибок необходимо повторять выполнение транзакции. Если по каким-либо причинам транзакция не может повторно выполниться после определенного количества повторений, то следует запустить компенсирующие транзакции и откатить все изменения.

Таким образом, решая проблемы межсервисной коммуникации, сервис-ориентированная архитектура становится очень сложной в разработке. Однако после ее построения, внедрение новых модулей в систему значительно упрощается [145].

Сервисы являются монолитными программами, что влияет на сложность сопровождения: чем больше нагружен сервис функциональностью, тем сложнее его изменять в дальнейшем.

Из-за того, что сервисы располагаются на разных вычислительных узлах и могут быть построены при помощи разных технологий, задачи тестирования и отладки сильно усложняются. Появляется необходимость создавать тестировочную среду, идентичную рабочей. Такая среда позволяет проводить интеграционные тесты всех модулей, а также моделировать произошедшие ошибки с целью их выявления без нарушения работы основной системы [146].

Разделение модулей на отдельные артефакты программы имеет множество преимуществ. Структура сервис-ориентированной системы наглядна и упрощает процесс документирования [147]. Сервисы просто развертывать ввиду того, что изменение одного сервиса не требует перекомпиляции остальных. Каждый сервис может быть реплицирован на различных вычислительных узлах, обеспечивая масштабируемость и отказоустойчивость системы [148].

Применение брокера сообщений реализует совместимость. Сторонние системы могут быть подключены к брокеру сообщений также, как обычный сервис системы [149].

Наиболее проблемная часть для сервис-ориентированной архитектуры является производительность [150]. На скорость отклика системы влияет задержка и потери сетевых запросов между вычислительными узлами, а также межсервисное взаимодействие через посредника.

Несмотря на возможные проблемы с производительностью, сервис-ориентированная архитектура часто применяется для создания распределённых систем.

Однако облачная среда отличается от корпоративной, предъявляя высокие требования к масштабируемости и отказоустойчивости. Публичные облачные системы должны обладать возможностью адаптироваться к непредсказуемой нагрузке. Помимо этого, публичная облачная система эволюционирует быстрее, чем корпоративная, из чего следует потребность в быстрой разработке и быстром развертывании. Более строгие требования к сервис-ориентированной архитектуре составили основу микросервисной архитектуры.

2.4 Адаптация сервис-ориентированной САПР обуви под облачную среду, используя микросервисный архитектурный шаблон

Микросервисная архитектура — это частный случай сервис-ориентированной архитектуры, с более строгими правилами построения систем [151]. Данная архитектура быстро завоевывает популярность в отрасли как жизнеспособная альтернатива монолитным системам и сервис-ориентированным архитектурам.

Сервисы в микросервисной архитектуре – автономны, независимы, и решают только одну задачу или часть задачи предметной области, что делает сервис очень маленьким [152].

Мнения о малых размерах сервисов разнятся. К примеру, в Amazon используется подход с «командами на две пиццы» (two pizzas team) [153]: в команде разработки микросервиса должно быть столько людей, чтобы их можно было накормить двумя пиццами. Джордж Ф. и Льюис Д. предлагают делить систему на сервисы так, чтобы один сервис мог полностью понять один человек [154]. Такой подход используется в Netflix [155]. В их облаке функционирует около 800 сервисов. Малый размер сервиса обеспечивает его быструю разработку, развертывание, и одновременно с этим позволяет полностью отказаться от сопровождения. Если в сервис необходимо внести значительные изменения, то его разрабатывают заново. Таким образом процесс разработки сервисов становится непрерывным, обеспечивая непрерывное развитие системы [156].

Микросервисная архитектура стала возможна благодаря технологии контейнеризации [157] - метод виртуализации на уровне операционной системы, при котором ее ядро поддерживает несколько изолированных экземпляров пространства пользователя вместо одного. С точки зрения пользователя эти экземпляры, контейнеры, полностью идентичны отдельной операционной системе. Ядро обеспечивает полную изолированность контейнеров, поэтому приложения из разных контейнеров не могут воздействовать друг на друга. В деактивированном состоянии контейнер – это двоичный файл. При активации контейнерный движок распаковывает из этого файла программы и библиотеки, передавая их ядру операционной системы. После запуска контейнер становится обычным системным процессом [158].

В отличие от аппаратной виртуализации, когда эмулируется аппаратное окружение и может быть запущено множество гостевых операционных систем, в контейнере эмулируется та операционная система, в которой был запущен контейнерный движок. При этом отсутствуют дополнительные накладные расходы ресурсов на эмуляцию виртуального оборудования и запуск полноценного экземпляра операционной системы, что характерно аппаратной виртуализации. Таким образом, контейнеры позволяют уместить гораздо больше приложений на одном физическом сервере, чем любая виртуальная машина, которая занимает гораздо больше аппаратных ресурсов. Контейнеризация позволяет также создавать портативное и целостное окружение для разработки, тестирования и последующего развертывания. В контексте микросервисной архитектуры, технология контейнеризации позволяет изолировать друг от друга микросервисы, запущенные на одном сервере [157].

Быстрое распространение и повсеместное применение технологии контейнеризации привело к необходимости их автоматизированного управления. Системы оркестрации — это программы для координации взаимодействия нескольких контейнеров, создания контейнерных кластеров, их гибкого развертывания и масштабирования. Оркестрация позволяет сократить трудозатраты на создание информационных системы из множества контейнеров [159].

Микросервисная архитектура создавалась с учетом использования описанных технологий. Каждый сервис располагается в своем собственном контейнере и оркестрируется в облачной платформе.

Быстрая разработка вместе с высокими нагрузками на сервис зачастую становятся причиной множества ошибок. Из-за этого микросервисная

архитектура предъявляет высокие требования к отказоустойчивости. В системе не должно быть «единой точки отказа» – компонента, при отказе которого, система приходила бы в нерабочее состояние [160].

В сервис-ориентированной архитектуре отказ брокера сообщений может нарушить работу всех сервисов одновременно. Чтобы избежать этого, в микросервисной архитектуре сервисы передают сообщения друг другу напрямую. При этом, чтобы не заниматься реализацией механизма предохранителя в каждом сервисе для обработки ошибки передачи запросов, все межсервисные взаимодействия проводятся асинхронно [161].

Отсутствие брокера сообщений возвращает проблемы регистрации сервисов, ошибок при передаче данных и балансировки нагрузки. Однако, в отличии от сервис-ориентированной архитектуры, эти задачи решаются при помощи технологий оркестрации [162]. Такие инструменты оркестрации, как Kubernetes, обладают виртуальной сетью, внутри которой функционируют контейнеры. Виртуальная сеть обладает своим собственным сервером доменных имен. Kubernetes вводит собственное понятие сервиса — абстракция, объединяющая множество одинаковых контейнеров. Каждому сервису дается свое имя, которое является доменным именем. Платформа позволяет контейнерам, подключенным к виртуальной сети, отправлять запросы, используя имена сервисов в URL. Таким образом платформа самостоятельно решает проблемы регистрации сервисов и балансировки нагрузки [163].

Применение средств платформы невозможно в гибридном облаке, что привело к появлению сервисной сетки (service mesh). Она реализует для группы сервисов такие функции, как мониторинг трафика, контроль доступа, обнаружение других сервисов и решает проблемы сквозной функциональности [164].

Работа сервисной сетки заключается в том, что к каждому контейнеру сервиса добавляется контейнер прокси-сервера, который контролирует весь входящий и исходящий из контейнера трафик. Такой прокси-сервер называется sidecar и может быть расположен внутри нескольких кластеров контейнеров, позволяя объединять несколько платформ и формируя гибридное облако [165].

Сервисная сетка предлагает к использованию реестр сервисов, который подключен к каждому прокси-серверу, что позволяет реализовать механизм межсервисного взаимодействия через обращение к доменному имени, что идентично методы платформы оркестрации [166].

Так как один и тот же прокси-сервер присоединяется ко всем сервисам, это позволяет внести в него общую для всех сервисов функциональность, включая задачи разграничения доступа, мониторинг активности и нагрузки и т.д.

Клиентские приложения или сторонние системы вынуждены работать с многочисленным количеством сервисов, и, при недостаточном доверии, могут не иметь возможности быть подключенным к сервисной сетке системы. Чтобы дать возможность использовать функционал системы извне, применяется шлюз программного интерфейса (API gateway) [167]. Это компонент, представляющий собой единую точку входа в систему, располагаясь между внешними системами и внутренними сервисами. Основная задача шлюза - маршрутизация, перенаправление запросов к конкретным сервисам, что позволяет скрыть внутреннюю структуру системы.

Также как и прокси-сервер сервисной сетки, шлюз может быть нагружен дополнительной функциональностью, необходимой для защиты системы. Рассмотрим подробнее технологии, используемые для защиты.

Все взаимодействия на транспортном уровне передачи данных как внутри, так и вне системы должны производиться с использованием TLS (Transport Layer Security) [168]. Сертификат TLS работает путём цифровой привязки криптографического ключа к идентифицирующей информации компании, что обеспечивает шифрование передачи данных. TLS работает, имея как частный, так и открытый ключ, а также отдельные ключи для каждого уникального безопасного сеанса. При первоначальном подключении общедоступные и закрытые ключи используются для создания ключа сеанса, к которому затем будут обращаться для шифрования и дешифрования передаваемых данных. Ключ сеанса действителен в течение ограниченного времени и предназначен только для данного сеанса.

В шлюзе программного интерфейса для защиты применяются техники ограничения частоты запросов, страхующие систему от искусственных высоких нагрузок. Известны две такие техники [169]:

- **Throttling -** предполагает, что входящие запросы принимаются не более одного раза в указанный период времени. Ограничения применяются отдельно для каждого соединения при резком росте нагрузки от этого соединения.
- **Debouncing -** предполагает, что все запросы будут игнорироваться до тех пор, пока они не прекратятся на определённый период времени. Только после этого запрос будет принят. Например, если установить таймер на 2 секунды, а запросы приходят 10 раз с интервалом в одну секунду, то фактическая обработка запроса произойдёт только спустя 2 секунды после последнего 10-го запроса. Такой метод также применяется отдельно для каждого соединения при резком росте нагрузки от этого соединения.

Обе техники одинаково эффективны [169] и выбор какой-то одной зависит от личных предпочтений разработчиков.

Проблема отказоустойчивости при высоких нагрузках проявляется в случае отключении одного компонента системы при сбое, от которого зависит работа множества других компонентов. Чтобы шлюз программного интерфейса не стал точкой единого отказа, необходимо создать множество статичных шлюзов, объединив их под одним доменным именем [170].

Единой точкой отказа является база данных. Чтобы устранить этот недостаток и обеспечить высокую отказоустойчивость, в микросервисной архитектуре каждый тип сервисов обладает своей собственной базой данных. Сервис хранит в ней только те данные, которые необходимы для работы. Если сервисам необходимы данные другого сервиса, то он выполняет соответствующий запрос [171].

Предлагаемая нами структура микросервисной САПР обуви изображена на рисунке 2.8.

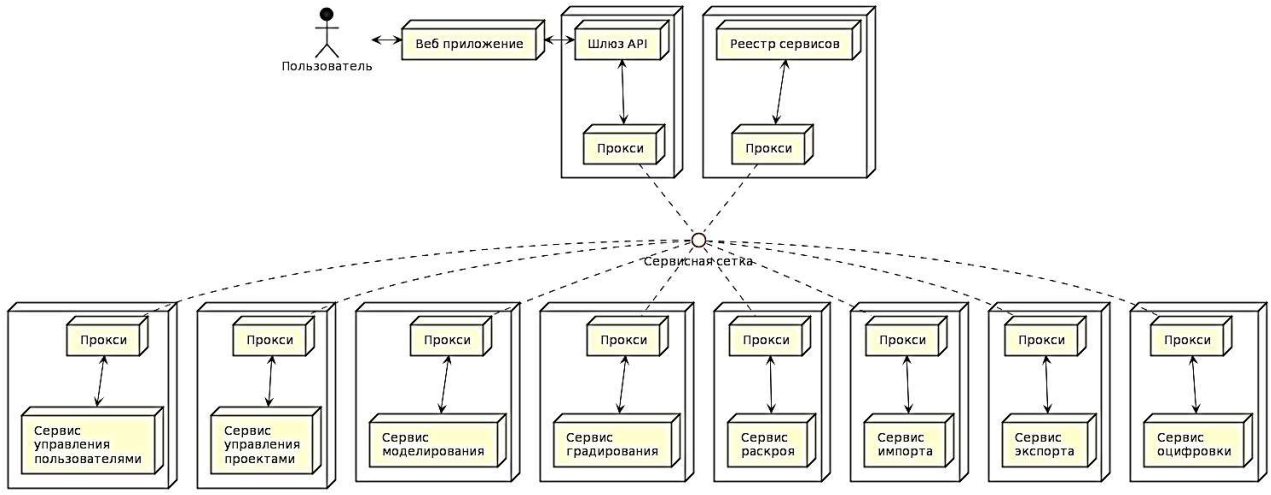


Рисунок 2.8 Микросервисная архитектура САПР обуви

Микросервисная архитектура лучше всего подходит для создания облачных систем. Разработка инфраструктуры хотя и сложна, однако

дальнейшее внедрение сервисов и их поддержка быстрее, чем в других архитектурах [160].

Тестирование и отладка сервисов, также как и в сервис-ориентированной архитектуре, требует построения тестировочной среды, идентичной рабочей [172].

Ввиду большого числа инфраструктурных компонентов, к новым разработчикам предъявляются высокие требования. Они должны знать об особенностях работы с платформой выполнения сервисов, шлюзом программного интерфейса и сервисной сеткой [160].

Микросервисная архитектура обладает наиболее эффективной развертываемостью, масштабируемостью, отказоустойчивостью и совместимостью [152].

Однако при этом, наиболее проблемное место для микросервисных систем – производительность [154]. Запрос от клиентского приложения передается в шлюз, затем в собственный прокси-сервер сервисной сетки, который направляет запрос в прокси-сервер сервиса, и только после этого доходит до самого сервиса. Запрос и ответ осуществляются в 8 пакетных прыжков, на каждом из которых имеется сетевая задержка и вероятность неконтролируемого сетевого сбоя.

Реализация облачной системы автоматизированного проектирования обуви с такой архитектурой возможна только при наличии больших нагрузок. Такая архитектура экономически затратна: каждый сервис состоит из 3-х элементов - основного контейнера, прокси-сервера и базы данных. Это делает всю систему более требовательной к вычислительным ресурсам. Рассмотрим возможности улучшения производительности облачной САПР обуви.

2.5 Разработка гибридной архитектуры облачной САПР обуви

Оценки проведенного анализа архитектурных паттернов представлены в таблице 2. Каждый из них обладает собственными достоинствами и недостатками, что делает их подходящими для конкретного сценария использования САПР обуви.

Таблица 2. Показатели анализа и оценки архитектурных паттернов.

Показатель	Архитектура			
	Монолитная	Микроядерная	Сервис-ориентированная	Микросервисная
	Оценка сложности			
Построение инфраструктуры	Низкая	Средняя	Высокая	Высокая
Разработка модулей	Высокая	Средняя	Средняя	Низкая
Сопровождение	Высокая	Средняя	Средняя	Низкая
Отладка и тестирование	Низкая	Высокая	Высокая	Высокая
Развертывание	Низкая	Низкая	Средняя	Средняя
Эффективность масштабируемости	Низкая	Средняя	Высокая	Высокая
Эффективность отказоустойчивости	Низкая	Средняя	Средняя	Высокая
Производительность	Высокая	Средняя	Низкая	Низкая
Интеграция со сторонними системами	Высокая	Высокая	Низкая	Низкая

В первой главе был проведен анализ САПР обуви, который показал, что системы встраиваются в процесс конструирования, а не определяют его. Это создает необходимость адаптации функционала системы отдельно под каждое предприятие. Разработчикам САПР обуви необходимо организовать процессы поддержки «непрерывных изменений», которые со временем приводят к

«битовой деградации» (bit rot) – разрушительном влиянии небольших изменений функционала на всю архитектуру. Для предотвращения этого необходимо обеспечить изменяемость не только отдельных частей САПР обуви, но и ее архитектуры [173].

В представленных ранее архитектурных паттернах встречается один и тот же элемент, который имеет разные названия и занимает разные позиции, но функционально выполняет одно и то же: ядро - в микроядерной архитектуре, брокер сообщений – в сервис-ориентированной и сервисная сетка - в микросервисной. Все эти элементы координируют модули системы. Отсюда появляется возможность сформировать такую архитектуру, которая могла бы изменять свою форму и масштабироваться за счет изменения только одного координирующего элемента [174]. Тогда, облачная САПР обуви сможет адаптироваться под различные условия эксплуатации, используя достоинства каждого архитектурного паттерна.

Рассмотрим базовую форму такой архитектуры – микроядерную (рис.2.9). Каждый модуль представляет собой сервис – автономный, независимый, и выполняющий только одну задачу предметной области, как в микросервисной архитектуре. Но взаимодействие производится через ядро, расположенное на том же вычислительном узле, что и модуль микроядерной архитектуры - путем передачи сообщений по модели издатель-подписчик. Это позволяет использовать более быстрые и надежные способы связи, чем сетевое взаимодействие.

Модули должны быть неактивны в основное время и запущены только для выполнения действия процесса обработки запроса. Ядро при этом реализует не только коммуникацию между модулями, но и всю сервисную

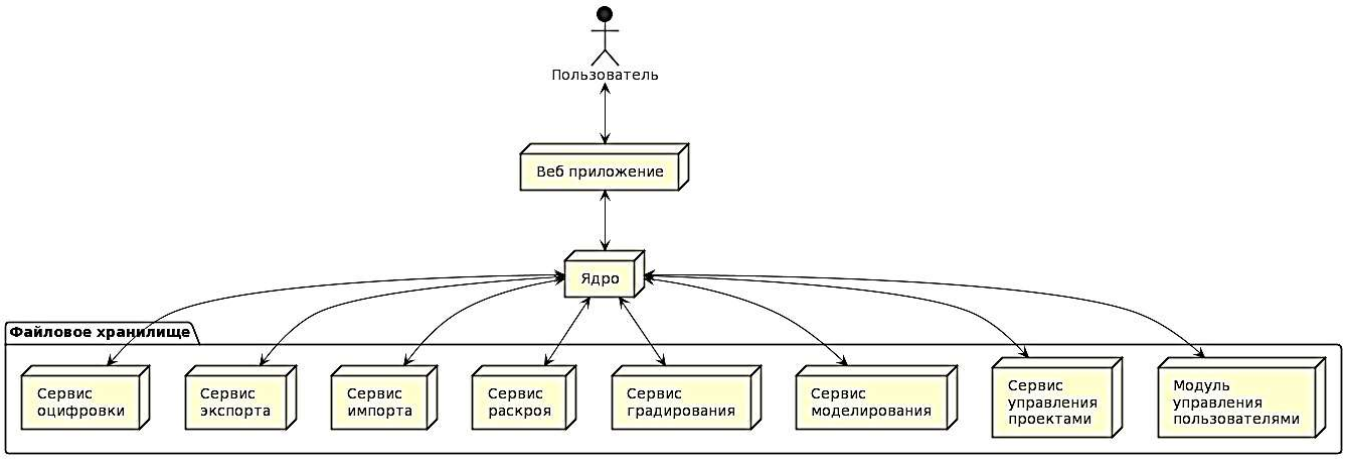


Рисунок 2.9 Микроядерная форма архитектуры облачной САПР обуви

сквозную функциональность - контроль доступа, мониторинг трафика, механизм предотвращения ошибок. Запуская модуль, ядро становится владельцем процесса модуля и позволяет отключить его в любой момент при аномальной активности. Таким образом, вычислительный узел содержит ядро и множество независимых программ-модулей. При обработке запроса ядро последовательно запускает модули, отслеживая результат их работы и ошибки. Развертывание новых сервисов и замена старых осуществляется заменой файлов в файловой системе вычислительного узла, которым также может быть и контейнер. Системы контейнеризации позволяют подключать к контейнеру постоянную файловую систему и работать с ней напрямую. Контейнер с ядром системы при инициализации копирует модули из постоянного файлового хранилища в свое временное хранилище. После чего может начинать работу.

Такая форма перенимает все положительные и отрицательные стороны микроядерной архитектуры, включая неэффективность масштабирования. Для масштабирования на небольшое количество вычислительных узлов, например, при построении частного облака предприятия, необходимо преобразовать архитектуру системы в сервис-ориентированную форму (рис.2.10).

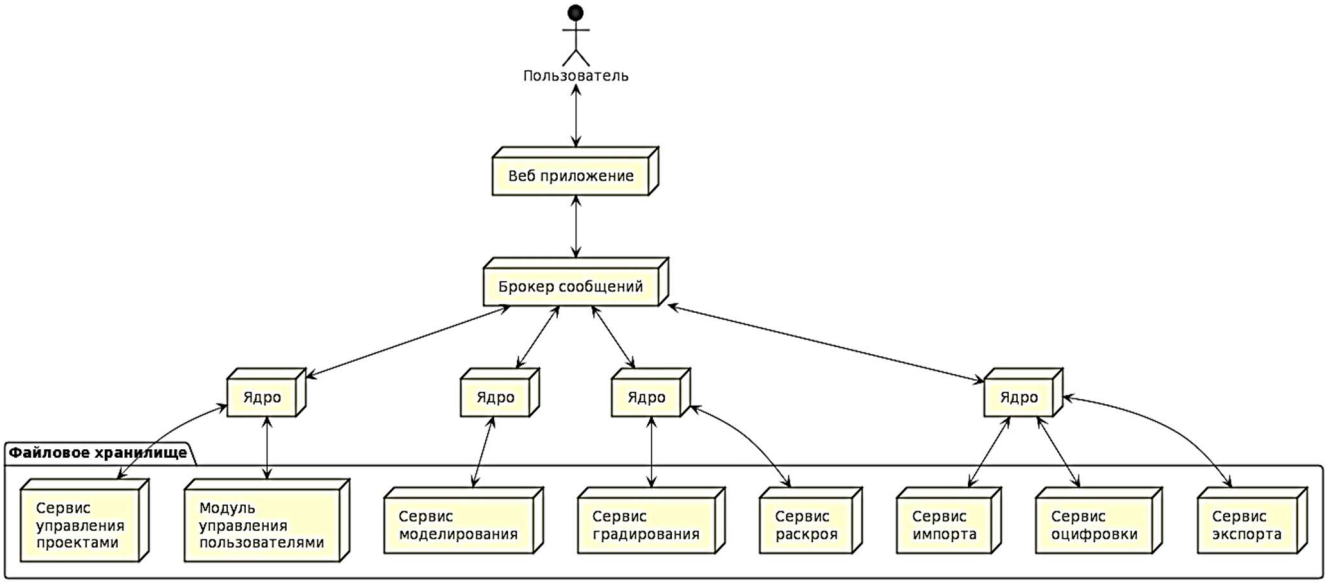


Рисунок 2.10 Сервис-ориентированная форма оптимизированной архитектуры облачной САПР обуви

Для этого следует развернуть сервисную шину и подключить к ней сервисы через ядро. В этом случае каждое ядро должно содержать собственный набор модулей, инкапсулировать их функциональность, формируя таким образом один большой сервис. Для еще большего масштабирования, например, при создании САПР в публичном облаке, необходимо преобразовать архитектуру системы в микросервисную форму (рис. 2.11).

Так ядро становится прокси-сервером сервисной сетки, который содержит и контролирует только один модуль.

Таким образом, способ коммуникации модуля и ядра никогда не меняется, что упраздняет необходимость изменения всех модулей системы для работы в новой форме архитектуры при осуществлении перехода. Рассмотрим подробнее коммуникацию между ядром и модулем.

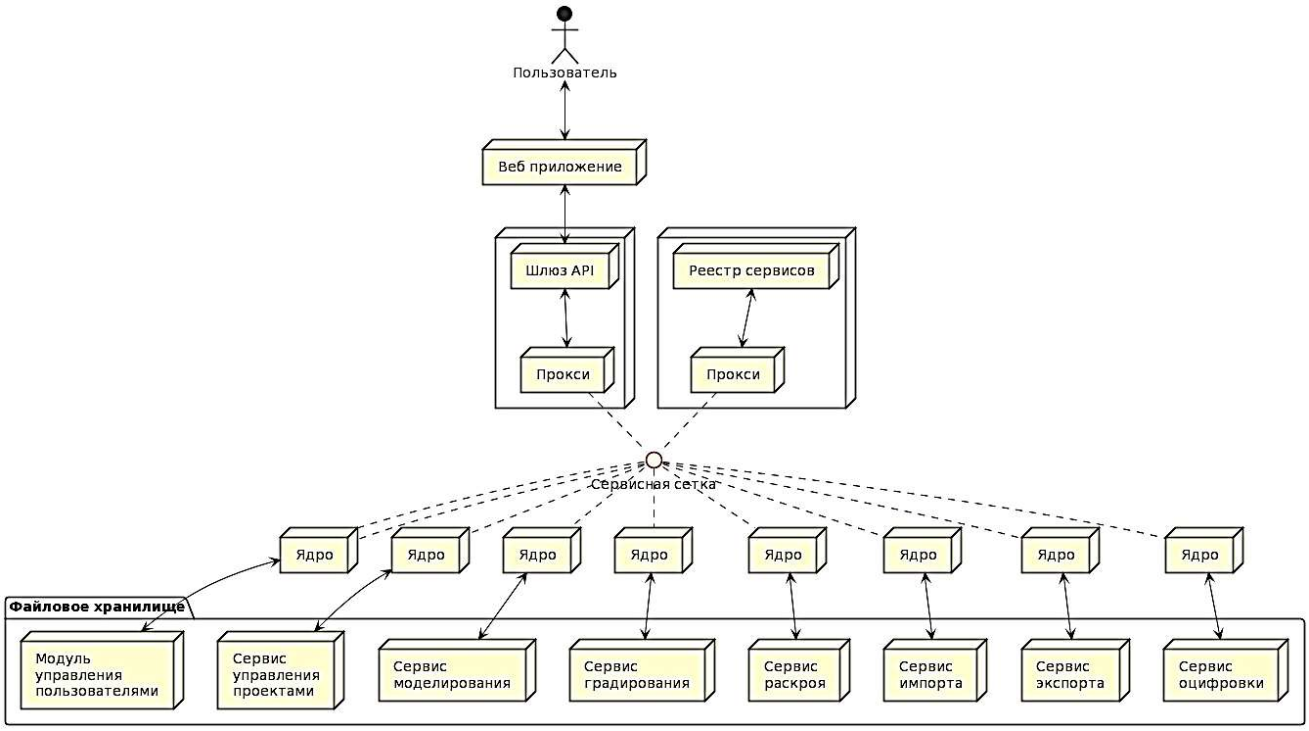


Рисунок 2.11 Микросервисная форма оптимизированной архитектуры облачной САПР обуви

Наиболее простой вид коммуникации — это стандартные потоки [175]. У каждой программы существует 3 системных потока:

- stdin - поток ввода данных;
- stdout - стандартный поток вывода данных для программ;
- stderr - поток вывода ошибок.

Потоки stdin и stdout могут иметь как текстовый, так и двоичный режим работы и не имеют ограничений по размерам данных. Распространенные веб-серверы используют эти потоки для выполнения CGI программ [176], что гарантирует жизнеспособность такой коммуникации.

Существует более быстрый способ обмена данными - отображение файла в память, когда файлу ставится в соответствие определённый участок оперативной памяти [177]. Любое обращение к файлу равносильно обращению

к оперативной памяти. Однако использование отображений влечет за собой замедления из-за страничных ошибок доступа. Например, страница, относящаяся к нужному файлу, уже находится в оперативной памяти, но не ассоциирована с файлом отображения. Если в этот момент часть файла будет изменена другим процессом, то попытка ассоциировать её с отображением может закончиться неудачей и привести к необходимости повторной записи данных [178]. Таким образом, операция записи данных может занять больше времени, чем стандартные потоки.

Другим способом взаимодействия между процессами являются каналы – неименованные (pipe) и именованные (fifo) [179]. С точки зрения процессов канал выглядит как два открытых файла – один на чтение и один на запись. Производить запись в канал можно до тех пор, пока есть место в буфере, если место в буфере кончится – процесс на записи будет заблокирован. Чтение возможно из канала пока есть данные в буфере, если данных нет – на чтении процесс будет заблокирован [180]. Размер буфера ограничен и устанавливается в конфигурации операционной системы кратным размеру страницы (4096 байт). Именованный канал FIFO доступен как объект в файловой системе. При этом до открытия объекта FIFO на чтение, собственно коммуникационного объекта не создаётся. После открытия объекта FIFO в одном процессе на чтение, а в другом на запись, возникает ситуация полностью эквивалентная использованию неименованного канала. Основное отличие между именованным и неименованным каналом - то, что неименованный может быть использован только процессами, находящимися в отношении родительский-дочерний, а именованный может использовать любая пара процессов [179]. Такой способ не подходит технологии контейнеризации, так как появляется необходимость настройки операционной системы.

В связи с этим для обмена данными между ядром и модулями рекомендуется использовать стандартные потоки, что обеспечивает максимальную производительность.

Масштабирование базы данных при смене формы архитектуры затруднительно. Разделение данных повлечет за собой изменение их модели и описания, что в свою очередь приведет к изменению всех модулей, работающих с базой данных. Поэтому необходимо сделать выбор: использовать ли одну базу данных, что повлияет на отказоустойчивость, или базу данных на каждый сервис отдельно, что потребует больших вычислительных ресурсов. В нашем случае рекомендуем использовать единую базу данных во всех архитектурных формах, так как применение многочисленных баз данных отдельно для каждого модуля экономически не эффективно. Помимо этого, отказоустойчивость, которую обеспечивает множество баз данных, заключается в том, что отдельная функциональность системы не будет работать на время сбоя. В САПР обуви отключение определенной функциональности нарушит работу всей системы, так как предметная область связывает модули логически. Например, для модуля градирования нужны данные модуля проектирования, а модулю раскроя необходимы данные модуля градирования. Сбой базы данных одного из модулей нарушает всю цепочку, что лишает смысла применение подхода с разделенными базами данных.

Разработанная архитектура в целом направлена на эффективную производительность, позволяя адаптировать систему под размеры предприятия с минимальными трудозатратами. Микроядерная форма подходит для частных облаков малых и средних предприятий, не требуя больших вычислительных ресурсов. Сервис-ориентированная форма подойдет для средних и крупных предприятий – процесс смены формы архитектуры будет заключаться в

разработке нового ядра и развертывании инфраструктурных компонентов. Малый размер сервисов обеспечивает минимальные трудозатраты по сопровождению. В случае необходимости внесения значительных изменений, сервис может быть полностью переписан.

В следующей главе диссертации демонстрируется реализация метода оцифровки в предложенной гибридной архитектуре облачной САПР обуви.

Выводы по второй главе

1. Рассмотрены понятия архитектуры, критерии, влияющие на процессы жизненного цикла системы, а также критерии качества. Показана необходимость модульного подхода при проектировании архитектуры облачной САПР обуви. Показано, что облачная САПР обуви должна включать модули: управления пользователями и проектами, моделирования, градирования, раскроя, импорта, оцифровки, экспорта.
2. Проведено проектирование архитектуры облачной САПР обуви на основе следующих шаблонов:
   - Монолитной архитектуры и его формах: трехуровневой и «чистой архитектуры». Анализ преимуществ и недостатков данного шаблона показал невозможность применения данного шаблона при проектировании облачной САПР обуви;
   - Микроядерной архитектуры. Проведен анализ его преимуществ и недостатков. Показано, что облачная САПР обуви может быть создана при помощи данного архитектурного шаблона, но его использование ограничивают проблемы, связанные с масштабированием системы;
   - Сервис-ориентированной архитектуры. Описаны элементы, применяемые для построения инфраструктуры: синхронный и

асинхронный метод взаимодействия компонентов; стили программного интерфейса REST и gRPC; применение брокера сообщений; распределенные транзакции. Проведен анализ преимуществ и недостатков такого шаблона. Показано, что облачная САПР обуви может быть построена в стиле сервис-ориентированной архитектуры. Инфраструктуру такой САПР сложно выстроить, но легко масштабировать. Выявлено, что сервис-ориентированный архитектурный шаблон испытывает проблемы при высоких нагрузках и требователен к вычислительным ресурсам, что отрицательно сказывается на стоимости содержания системы.

— Микросервисной архитектуры, как частный случай сервис-ориентированной архитектуры, обладающей отказоустойчивостью. Показано, что данная архитектура лучше всего подходит для создания облачной САПР обуви. Однако она слишком требовательна к вычислительным ресурсам, что отрицательно сказывается на стоимости содержания системы.

3. Предложена гибридная архитектура облачной САПР обуви, которая в качестве компонентов содержит модель ядра микроядерной архитектуры, модель модуля микросервисной архитектуры, модель взаимодействия сервис-ориентированной архитектуры. Показано, что изменение ядра системы и развертывание инфраструктурных компонентов позволяет с минимальными трудозатратами изменить общую форму архитектуры.

3. ДЕТАЛИЗАЦИЯ КОМПОНЕНТОВ ОБЛАЧНОЙ САПР ОБУВИ ГИБРИДНОЙ СТРУКТУРЫ

3.1 Разработка ядра облачной САПР обуви

Ядро занимается обменом сообщений между сервисами и обеспечивает сквозную функциональность. Рассмотрим подсистемы, которые составляют ядро.

CoreEngine – движок ядра, обеспечивающий запуск и работу ядра. Он связан с базой данных, в которой регистрирует себя и все подключенные сервисы; периодически опрашивает базу данных для обнаружения сервисов на других вычислительных узлах; следит за ресурсами вычислительного узла и контролирует его нагрузку. CoreEngine может работать в двух режимах – главном и ведомом. В главном режиме CoreEngine координирует запуск сервисов на других вычислительных узлах, что необходимо для сервис-ориентированной и микросервисной архитектурных форм. Отдельные подсистемы в главном режиме выполняют действия, требующие отсутствия конкуренции и параллелизма. В базе данных напротив главного ядра стоит соответствующий флаг с датой последнего опроса. Опрашивая базу данных о внутренней структуре системы, ядро обновляет дату. Если с момента последнего опроса прошло больше времени, чем установлено лимитом системы, то роль главного ядра на себя берет следующее ядро по списку в базе данных. Ведомое ядро при запуске для получения списка сервисов, которые следует активировать, опрашивает главное.

ConfigContainer – подсистема, предоставляющая унифицированный доступ к конфигурации ядра. Конфигурация представляет собой множество пар ключ-значение, которые могут использовать любая подсистемы ядра или

модули. Физически конфигурация может храниться разными способами, например, через текстовый файл или переменные среды. Задача данной подсистемы – получить это множество значений, загрузить в память и сообщать компонентам значения по заданному ключу.

LocalRepository – подсистема, контролирующая локальные временные файлы. Данные файлы хранятся на вычислительном узле, где располагается ядро. Соответственно, только ядро и модули этого вычислительного узла имеют доступ к локальным временным файлам. Такие файлы необходимы при одноразовой обработке файлов, загружаемых пользователем. Например, к таким файлам можно отнести проекты, импортируемые из других систем. Веб-системы не осуществляют такой контроль действий пользователей, как в десктопных системах. Так, если пользователь, начавший операцию импорта и загрузивший файл, закрыл браузер или соединение с ним прервано, то об этом событии сервер не получит информацию. Тогда, загруженный пользователем файл не будет удален и останется в файловой системе вычислительного узла на неопределенный срок. Чтобы неуничтоженные объекты не занимали место, необходимо сгруппировать файлы и папки, относящиеся к определенной операции, и назначить группе время жизни. По истечении этого времени группа должна быть уничтожена. Чтобы не нагружать ядро периодическими проверками, следует использовать «ленивое» сканирование. Время жизни каждого объекта проверяется только тогда, когда к какому-либо временному файлу идет обращение. Например, во время загрузки нового файла, подсистема LocalRepository параллельно должна проверить и удалить группы, время жизни которых истекло. Модули могут получить список таких групп, а также информацию об их содержимом. Для работы с локальными временными файлами подсистема LocalRepository может выдать модулю путь к нужной

группе файлов. Работа с файлами и папками внутри группы, включая создание и удаление, не ограничена для модуля.

GlobalRepository – подсистема, контролирующая временные файлы системы. Иногда работа с временными файлами осуществляется в несколько запросов и использование горизонтального масштабирования не гарантирует пользователю работу с одним и тем же вычислительным узлом. Соответственно, появляется задача предоставить доступ всем вычислительным узлам к единому хранилищу временных данных. Физически таким хранилищем выступает распределенная файловая система – облачное хранилище. Работа GlobalRepository с облачным хранилищем осуществляется аналогично LocalRepository. Исключением является очистка хранилища, так как в этом процессе может быть конкуренция. Очисткой хранилища занимается только тот GlobalRepository, CoreEngine которого находится в главном режиме. Связь с облачным хранилищем более сложная, чем работа с обычной файловой системой вычислительного узла. Поэтому модули не имеют прямого доступа к группе файлов в облачном хранилище. Вместо этого, подсистема GlobalRepository должна осуществлять загрузку группы временных файлов в файловую систему вычислительного узла, оставляя их под контролем LocalRepository. Аналогично этой операции, любая локальная группа временных файлов может быть перемещена в облачное хранилище. Например, пользователь загружает файл, модуль проводит над ним первичную обработку, отправляет пользователю ответ и сообщает ядру, что файл необходимо сохранить в общем хранилище временных данных, так как обработка не была закончена. Следующим запросом, пользователь может попасть на совершенно другой вычислительный узел, который загрузит в свою файловую систему необходимый файл из облачного хранилища и проведет оставшийся этап обработки.

ServicesRepository – подсистема, хранящая данные о всех возможных сервисах системы. Все сервисы расположены в облачном хранилище. Каждый сервис определяется по директории, внутри которой расположен файл с метаданными, описывающими сервис. Также, внутри директории расположены все артефакты сервиса – исполняемые файлы и библиотеки. При инициализации подсистема ServicesRepository должна просканировать облачное хранилище, найти сервисы по файлам с метаданными и сформировать локальную базу данных всех видов сервисов. Данная подсистема имеет возможность загрузить какой-либо сервис из облачного хранилища в файловую систему вычислительного узла.

InternalServicesRegistry – подсистема, регистрирующие внутренние сервисы в ядре. При инициализации InternalServicesRegistry обращается к CoreEngine и получает список запускаемых сервисов. Затем для загрузки нужных сервисов в файловую систему вычислительного узла она обращается к ServicesRepository. После этого проверяется работоспособность загруженных сервисов, для этого все сервисы должны обрабатывать специальный параметр запуска. Такой параметр не должен запускать какие-либо процессы обработки, а запущенный сервис должен сразу без ошибок завершить свою работу. Данная проверка позволяет определить совместимость сервиса с операционной системой и другими платформами выполнения программ.

ExternalServicesRegistry – подсистема, аналогичная InternalServicesRegistry, но хранящая данные по сервисам на других вычислительных узлах. Вместо сканирования, она получает данные напрямую от движка CoreEngine, который опрашивает базу данных для получения информации о структуре всей системы.

ServiceHandler – подсистема, запускающая сервис, и контролирующая его ресурсы. Пока сервис работает, ServiceHandler проверяет загруженность процессора запущенным процессом и размер занимаемой им памяти. Если сервис занимает слишком много ресурсов, то подсистема отключает запущенный процесс. На протяжении всей работы, ведет отчет для CoreEngine.

RequestHandler – основная подсистема ядра, которая обрабатывает запросы, принимает сообщения от сервисов и передает их в другие сервисы. Для осуществления этой задачи, RequestHandler обращается к ServiceRepository и находит тип сервиса, который может обработать сообщение. Затем обращение идет к InternalServicesRegistry и ExternalServicesRegistry, чтобы найти местоположение нужного сервиса. Если сервис находится на том же вычислительном узле, то сообщение передается в ServiceHandler, который запускает нужный сервис и передает ему данные. Если же нужный сервис находится на другом вычислительном узле, то сообщение отправляется через CoreServer далее к нужному сервису. Помимо этого, RequestHandler контролирует статичные ресурсы системы: изображения, веб-страницы и т.д. Каждый сервис может обладать набором таких ресурсов, и доступ к ним осуществляется через URL, а не через сообщения.

RequestQueue – подсистема, временно удерживающая запросы во время высоких нагрузок.

CoreServer – подсистема, принимающая и отправляющая запросы по сети, необходима для взаимодействия с пользователем и ядрами на других вычислительных узлах.

CoreLogger – подсистема создания и хранения логов. В данную подсистему попадает информация о каждом действии, его результате из других подсистем. Logger может хранить как общую информацию, так и осуществлять

трассировку. Под трассировкой понимается последовательность действий системы от начала запроса до появления ошибки. Соответственно, если ошибка отсутствует, то информация не сохраняется. Хранить логи – файлы с действиями системы, можно как на вычислительном узле, так и в облачном хранилище. Это зависит от характера хранения локальных данных. Например, если вычислительным узлом является контейнер, то все данные внутри него будут уничтожены сразу после его отключения. Соответственно, для контейнерной среды необходимо хранить логи в облачном хранилище.

CoreMonitor – подсистема, занимающаяся подсчетом метрик системы, таких как: количество успешных запросов, количество ошибок, средняя и максимальная нагрузка на процессор, средняя и максимальное значение занимаемой оперативной памяти, и т.д.

CoreAuth – подсистема, выполняющая контроль доступа к сервису.

Рассмотрим подробнее механизм контроля доступа в системе. Аутентификация – это процесс проверки подлинности идентификационных данных [181]. В облачных системах наиболее распространен механизм аутентификации на основе токенов. Данный метод позволяет пользователю самостоятельно предоставлять все необходимые данные вместе с запросом, подлинность которых гарантирует цифровая подпись. После входа в систему сервис аутентификации получает всю необходимую информацию о пользователе из базы данных, записывает ее в специальную строку - токен. Затем, данный токен подписывается электронной подписью и выдается пользователю. Таким образом токен, приложенный к запросу, позволяет идентифицировать отправителя. В токене хранится вся информация. Существуют разные форматы токенов, обеспечивающих разный уровень защиты. Наибольшее распространение получил формат Json Web Token (JWT), состоящий из нескольких частей. Заголовок (header) содержит общую

информацию о токене, а также кодовое название алгоритма вычисления цифровой подписи. Полезные данные (payload) содержат информацию для идентификации пользователя и его роли. Цифровая подпись (signature) создается следующим образом: заголовок и полезные данные конвертируют в формат base64 – текстовое представление двоичных данных. Такой формат позволяет токену быть независимым от кодировки символов. Сконвертированные заголовок и полезные данные записывают в строку через разделитель – точку. К полученной строке применяют HMAC версию хэш-функции, например: HMAC-SHA256, HMAC-SHA1, HMAC-MD5, HMAC-RIPEMD128 и другие. Полученная в результате хэш-строка является цифровой подписью, гарантирующей подлинность данных токена [182] (рис.3.1).

Рисунок 3.1 Анатомия токена Json Web Token

Главное достоинство аутентификации на основе токенов является его существенным недостатком. Самодостаточность токенов позволяет отказаться от связи с общей базой данных пользователей, усилив автономность системы, а также ускорить процесс аутентификации. Однако при этом система лишается контроля над пользователями. Если пользователь захочет выйти из системы

или токен был скомпрометирован, то у системы нет возможности отозвать его. Решением этой проблемы является выдача токена на очень короткий срок и создание системы повторной его выдачи без действий пользователя.

Такой механизм очень сложный и его реализация оправдана при разделении данных на множество баз данных для каждого сервиса отдельно. Так как нами решено использовать единую базу данных для всей системы, то наиболее подходящим механизмом является аутентификация, основанная на сессиях. Каждая сессия хранится в базе данных, что потребует от ядра системы осуществить поиск нужного пользователя по базе данных. Это скажется на производительности системы, но обеспечит лучшую безопасность [183].

Общая схема работы сессионной аутентификации описывается следующим процессом. После входа пользователя система генерирует случайную строку – идентификатор сессии, который выдается пользователю и одновременно с этим записывается в базу данных сессий. При каждом запросе пользователя, система достает из заголовка HTTP запроса идентификатор сессии и обращается к базе данных, сравнивая строки. Найденная запись в базе данных может содержать всю необходимую для работы информацию о пользователе. Для улучшения механизма аутентификации в базу данных можно добавить записи о IP адресе, с которого осуществлялся вход пользователя в систему, и данные о браузере – строку UserAgent. Таким образом, если злоумышленник похитит идентификатор сессии, то он не сможет отправлять запросы от системы пользователя без использования вирусных программ [184].

Рассмотрим основные данные, которые задействованы в работе инфраструктуры.

Каждый модуль содержит файл с метаданными, в котором записано кодовое название сервиса и его версия. Эти значения записываются в базу

данных при регистрации сервиса в системе. Также, в файле с метаданными указан путь к запускаемому файлу сервиса и список аргументов, которые необходимо передать при обычном запуске сервиса. Чтобы ядро могло координировать сообщения, в файле с метаданными должны быть перечислены все виды сообщений, которые принимает сервис. Для каждого принимаемого сообщения указывается его кодовое название. Одно и то же сообщение может поступать в ядро при выполнении разных операций. Поэтому для уточнения можно указать кодовое название выполняемой операции action. Чтобы совсем ограничить круг принимаемых сообщений, можно указать кодовое название сервиса отправителя sender. Некоторые сообщения могут быть обработаны пользователем без контроля доступа. Это поведение регулирует поле auth, содержащее кодовые названия групп пользователей. Также, для каждого сообщения можно указать аргументы запуска сервиса arguments, которые добавляются к указанным раннее аргументам. При обработке запроса сервису могут понадобиться технические данные о запросе, например, в заголовке HTTP запроса передается информация о языке пользователя. Чтобы ядро передавало эти данные, в описании сообщения присутствует флаг addRequest. По аналогии, механизм аутентификации ядра способен передать информацию о пользователе, если будет указан флаг addClaims. Помимо сообщений, сервисы могут обладать статичными ресурсами: изображениями, веб-страницами и т.д. Они также перечислены в файле с метаданными. Каждый ресурс имеет URL, для обращения, и путь к ресурсам в файловой системе вычислительного узла. Также, как и сообщения, ресурсы могут быть защищены от несанкционированного доступа, за что отвечает поле auth. Представить файл с метаданными можно в различных форматах, но наиболее распространенным форматом является JSON.

JSON (JavaScript Object Notation) – это формат для хранения и обмена информацией, который одновременно хорошо обрабатывается компьютером, и нагляден для человека. JSON представляет собой строку, в которой описаны структура объекта и его свойства на языке программирования JavaScript [185], пример метаданных сервиса приведен на рисунке 3.2.

```
{
  "service": {
    "name": "UsersService",
    "version": "1.0.0",
    "executable": "users.exe",
    "arguments": "-production"
  },
  "messages": [
    {
      "name": "getUsers",
      "operation": "*",
      "sender": "*",
      "auth": "system",
      "arguments": "-service",
      "addRequest": "false",
      "addClaims": "false"
    },
    {
      "name": "getUsers",
      "operation": "*",
      "sender": "*",
      "auth": "admin, support",
      "arguments": "-admin",
      "addRequest": "true",
      "addClaims": "true"
    }
  ],
  "resources": {
    "resource": {
      "url": "/users",
      "path": "/content/*",
      "auth": "admin, support"
    }
  }
}
```

Рисунок 3.2 Пример метаданных сервиса в формате JSON

Сообщения, которые передают друг другу сервисы, также имеют формат JSON. Данный формат отлично подходит для передачи данных по сети.

Сообщение содержит несколько объектов. Заголовок header, включающий идентификатор запроса id, кодовые названия сообщения name, выполняемой операции action и сервиса-отправителя sender. Так как взаимодействие с пользователем по протоколу HTTP является односторонним, сервису необходимо сразу отправить ответ на запрос.

Флаг sync указывает подсистеме RequestHandler, что соединение необходимо удерживать до конца обработки запроса. Объект data содержит основную информацию, необходимую для выполнения задачи сервисом. Сообщение в таком виде предается между сервисами по сети. Внутри ядра данное сообщение может быть дополнено подсистемами. Например, CoreServer дополняет сообщение объектом request, содержащим информацию о HTTP запросе – заголовки, IP адрес отправителя и т.д. Из дополненной информации о запросе CoreAuth может получить данные для процесса аутентификации. Получив из базы данных информацию о пользователе, который отправил запрос, CoreAuth добавляет к сообщению объект claims, с данными о пользователе. Затем, полное сообщение передается в сервис для обработки. Пример такого сообщения показан на рисунке 3.3.

Рассмотрим различные сценарии работы ядра.

Получение обычного запроса. CoreServer получает запрос, в заголовке которого указан content-type равный application/json. Создается поток ожидания обработки запроса. Запрос передается в RequestHandler, который распаковывает сообщение из запроса и дополняет его данными из заголовка запроса. Потоку ожидания обработки сообщения присваивается идентификатор сообщения. Из сообщения получают названия: сообщения, действия и отправителя (поля name, action, sender). Полученные значения передаются в InternalServiceRegistry и ExternalServiceRegistry параллельно, для поиска сервисов, способных

```
{
  "message": {
    "header": {
      "id": "fa37864e-53e3-4eea-8d5d-42de31c03c77",
      "name": "getUsers",
      "action": "ACP_USERS",
      "sender": "ACP",
      "sync": "false",
      "data": {
        "fromId": 1,
        "toId": 300,
        "fields": [
          "id",
          "name"
        ]
      },
      "request": {
        "url": "/",
        "connection": "192.168.0.1",
        "header": [
          {
            "key": "accept",
            "value": "json"
          },
          {
            "key": "lang",
            "value": "russian"
          },
          {
            "key": "session",
            "value": "B10A8DB164E0754105B7A99BE72E3FE5"
          }
        ]
      },
      "claims": {
        "userId": 1,
        "userName": "admin",
        "userGroup": "admin",
        "companyId": "",
        "companyName": ""
      }
    }
  }
}
```

Рисунок 3.3 Пример полного сообщения запроса списка пользователей системы в формате JSON

обработать сообщения. Если сервисы не найдены – обратно отправляется сообщение об ошибке. Если на данном вычислительном узле нет нужного сервиса – InternalServiceRegistry вернул пустой список, тогда оригинальный запрос вместе с адресом вычислительного узла, обладающего нужным сервисом) передается в CoreServer, который отправляет запрос со статусом 303 (перенаправление) на переданный адрес. В случае, если вычислительный узел обладает сервисом, способным обработать запрос, проверяется флаг

синхронной обработки сообщения (поле sync) и выполняется обработка синхронного или асинхронного запроса.

Получение запроса с файлом. CoreServer получает запрос, в заголовке которого указан content-type равный multipart/form-data. Создается поток ожидания обработки запроса. Из двоичных данных получает объекты message и file. CoreServer передает объект file в LocalRepository, который создает новую запись о временном файле, сохраняет содержимое file и возвращает информацию о записи. CoreServer, конвертирует объект message в текстовый формат и передает его в RequestHandler вместе с информацией о временном файле. RequestHandler распаковывает сообщение из запроса и дополняет его информацией о временном файле, а также информацией из заголовка запроса, присваивает потоку ожидания идентификатор сообщения. Последующие действия идентичны процессу получения обычного запроса.

Синхронная обработка сообщения. RequestHandler запрашивает данные о загруженности вычислительного узла у CoreEngine. В случае, если узел нагружен выше лимита, установленного конфигурацией, то производится перенаправление запроса или возврат ошибки по описанному раннее алгоритму. Если узел не нагружен, то выполняется аутентификация.

Асинхронная обработка сообщения. RequestHandler передает в CoreServer информацию о том, что сообщение принято. После чего, CoreServer отправляет ответ на запрос со статусом 200 (успешно), соединение с отправителем закрывается и поток ожидания уничтожается. Само сообщение помещается в очередь RequestQueue. Каждый раз, когда нагрузка на вычислительный узел ниже лимита, установленного конфигурацией, CoreEngine передает сообщение из очереди в RequestHandler и начинается процесс аутентификации.

Аутентификация. RequestHandler передает сообщение и информацию о сервисе в CoreAuth. Из заголовка запроса, которым было дополнено сообщение при помощи CoreServer, CoreAuth получает идентификатор сессии, IP адрес отправителя и данные о браузере. Если в сообщении отправитель указан, как user, то полученные данные отправляются в базу данных для поиска сессии. Если сессия не найдена, то в RequestHandler возвращается ошибка, и начинается процесс формирования сообщения об ошибке. Если сессия найдена, то информация о ней дописывается в сообщение, которое возвращается в RequestHandler. Любое другое значение отправителя, кроме user, говорит о том, что сообщение было отправлено другим вычислительным узлом, а не пользователем. В таком случае, значение идентификатора расшифровывается алгоритмом AES, используя ключ, указанный в конфигурации системы. Полученное значение является идентификатором узла-отправителя в базе данных. После проверки наличия полученного значения в базе данных, информация об отправителе запроса не дописывается к сообщению. Возвращенное в RequestHandler сообщение приводит к запуску сервиса.

Запуск сервиса. RequestHandler передает полное сообщение и информацию о запускаемом сервисе в ServiceHandler, который запускает процесс, с указанными в информации о сервисе аргументами, и перенаправлением стандартных потоков. В поток stdin подается сообщение. Из потока stdout читается результат обработки. Во время выполнения обработки, в поток stdout могут подаваться команды для ядра, например перемещение файла из локальной файловой системы в облачное хранилище. Каждая команда представляет собой строку вызова функции: сначала идет название функции, затем в скобках аргументы через запятую и концом команды является разделитель – точка с запятой. Специальная команда exit завершает работу сервиса, в параметрах команды указывается ответное сообщение в формате

JSON. Завершение работы сервиса без этой команды расценивается как ошибка. При отсутствии результирующего сообщения работа по обработке запроса заканчивается. В ином случае формируется ответ.

Отправка сообщения. В RequestHandler передается сообщение. Если в нем выставлен флаг асинхронной обработки, то оно отправляется в RequestQueue и обрабатывается также как и входящие сообщения. При синхронной обработке сообщения, предполагается, что ведется обработка ответного сообщения, и идентификатор ответа соответствует идентификатору запроса. В таком случае, сообщение упаковывается в формат протокола HTTP. Если в сообщении были указаны данные по файлу, то все данные конвертируются в двоичный формат и дополняются файлом, запись в LocalRepository удаляется. Затем ответ передается в CoreServer, который ищет поток ожидания обработки с идентификатором, равным идентификатору сообщения. После чего, ответ отправляется по открытому соединению и поток уничтожается. Если поток не найден, значит обработка заняла слишком много времени или соединение прервалось из-за проблем с сетью. CoreRequest возвращает ответ в RequestHandler для формирования отложенного сообщения.

Формирование отложенного сообщения. RequestHandler распаковывает сообщение, проверяет наличие файла. Сохраняет файл в LocalRepository, после чего переносит запись из LocalRepository в GlobalRepository, тем самым зафиксировав данные в облачном хранилище. Запись будет иметь новый идентификатор, поэтому RequestHandler изменяет информацию о файле в сообщении. После чего, загружает сообщение в базу данных.

Получение отложенного сообщения. Предполагается, что если отправитель не получил данные синхронным запросом, как ожидал, то через какое-то время он повторит запрос с флагом retry. При распаковке сообщения,

наличие этого флага будет означать обработку отложенного сообщения. Для этого RequestHandler найдет сервис, который должен его обрабатывать. Без поиска сервиса на вычислительных узлах, RequestHandler должен провести аутентификацию сообщения и при наличии права доступа обратиться к базе данных. Для поиска отложенного сообщения в базе данных идентификатор повторного сообщения должен совпадать с идентификатором оригинального сообщения. RequestHandler загружает сообщение и проверяет наличие файлов. Если в теле сообщения присутствует запись о файле, то производится его поиск в GlobalRepository. Найденный файл загружается в LocalRepository, и затем производится стандартная отправка ответного сообщения синхронным методом.

Рассмотренные сценарии являются примером возможной реализации ядра, которую необходимо скорректировать под определенную вычислительную нагрузку. Например, для высоконагруженных систем процесс обработки следует оптимизировать, использовав технологии кэширования.

Определив форму передаваемых сообщений и порядок работы ядра, перейдем к разработке метода оцифровки в модуле облачной САПР обуви.

3.2 Разработка метода оцифровки в модуле облачной САПР обуви

В первой главе нами рассмотрена проблема оцифровки в САПР обуви. Существующие методы оцифровки неэффективны. Применение дигитайзера требует дополнительных экономических затрат. В свою очередь, оцифровка по изображению производится с лекалами. Для получения чертежа с изображения необходимо применить более сложные алгоритмы, требовательные к вычислительным мощностям.

Представим метод оцифровки, использующий возможности облачных технологий. Весь процесс можно разделить на 3 этапа: получение изображения, предварительная обработка и векторизация.

Для получения изображения нами предложено использовать одно из самых распространенных вычислительных устройств [186] – смартфон. Аналитическая компания Strategy Analytics выпустила отчет, согласно которому глобальная база пользователей смартфонов резко выросла с 30 тысяч человек в 1994 году до 1 миллиарда в 2012 году и рекордных 3.95 миллиарда в июне 2021 года. В общей сложности в июне 2021 года население земного шара насчитывало приблизительно 7.90 миллиарда человек, то есть половина всех людей уже владела смартфонами. Смартфоны стали незаменимым инструментом в повседневной жизни и по прогнозам аналитиков к 2030 году во всем мире их будут использовать уже 5 миллиардов человек [187]. Смартфоны редко выпускаются без встроенной камеры, и улучшение камеры – это один из основных векторов развития множества компаний-производителей смартфонов. О качестве получаемых изображений говорит событие, произошедшее в середине 2020 года. Японская компания Olympus сообщила о продаже своего фотобизнеса компании Japan Industrial Partners (JIP), что означало уход с рынка одной из крупнейших компаний по производству фототехники. Из-за роста популярности смартфонов, в которые устанавливают качественные камеры, продажи Olympus снизились, что привело к серьезным убыткам компании в течение трех лет [188]. Таким образом, для проведения операции оцифровки камера смартфона подходит не только по техническим качествам, но и экономическим соображениям.

Полученное изображение необходимо подготовить к векторизации, для этого требуется устранить искажения, шумы, и лишние объекты, не относящиеся к чертежу [189]. Помимо этого, изображение нужно обязательно

привести к реальным размерам. Этот этап выполняется при помощи средств технического зрения, из которых наиболее функциональным и распространенным является библиотека OpenCV (Open Source Computer Vision Library). Эта библиотека распространяется с лицензией BSD и открытым программным кодом. Она постоянно дорабатывается, поддерживает разные программные языки, включая C/C++, Java, Phyton. На сайте библиотеки доступны версии для операционных систем Windows, Mac OS, Linux, Android и IOS. Имеется подробная документация, примеры, множество форумов и книг по использованию OpenCV. Спектр функциональных возможностей библиотеки обширен: обработка и очистка изображений, распознавание образов, захват видео, калибровка камер, применение морфологических оператор, работа с матрицами и др. Примечательно, что разработчиком библиотеки является Российская компания Itseez [186].

Рассмотрим алгоритм подготовки изображения с использованием функций библиотеки OpenCV.

Калибровка камеры. Теоретически можно создать линзу, которая не вносит никаких искажений. Но на практике идеальных линз не бывает. Связано это с особенностями производства: дешевле изготовить «сферическую» линзу, чем математически правильную «параболическую». Кроме того, трудно механически совместить линзу с блоком формирования изображения. Рассмотрим два основных вида искажений (дисторсий) объектива [186]. Радиальная дисторсия (рис. 3.4.а) является результатом неправильной формы линзы: объектив камеры вносит заметные искажения по краям формирователя изображения. Это выгибание является причиной эффекта «бочки», или «рыбьего глаза». Лучи, проходящие дальше от центра объектива, изгибаются сильнее тех, что проходят ближе к центру. В оптическом центре формирователя изображения искажение равно 0 и увеличивается по мере приближения к

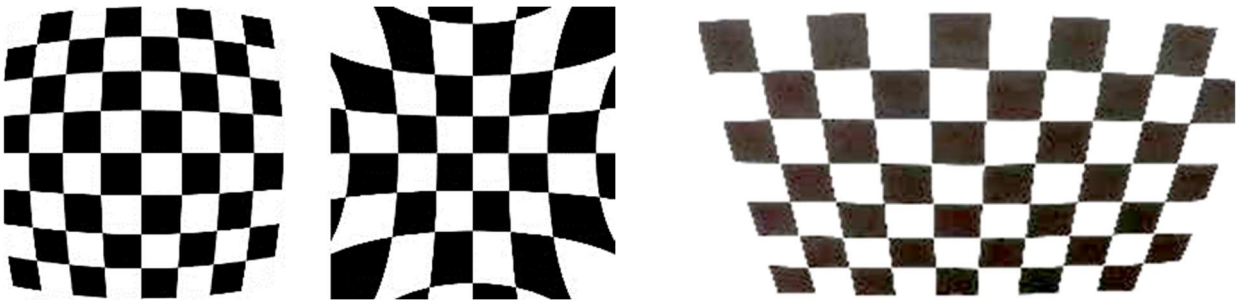
периферии. В общем случае радиальное положение точки на плоскости снимка корректируется по следующим формулам [190]:

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$r^2 = x^2 + y^2,$$

где (x, y) – исходные координаты точки на плоскости снимка, $(x_{corrected}, y_{corrected})$ – скорректированные, k_1, k_2, k_3 – коэффициенты корректировки радиальной дисторсии.

Второй распространенный вид дисторсии – тангенциальная (рис 3.4.б), вызванная дефектами производства, из-за которых линза не строго параллельна плоскости снимка. Тангенциальная дисторсия корректируется коэффициентами p_1, p_2 [190]:

$$x_{corrected} = x + 2p_1 xy + p_2(r^2 + 2x^2)$$
$$y_{correcte} = y + p_1(r^2 + 2y^2) + 2p_2 xy$$
$$r^2 = x^2 + y^2$$



а б

Рисунок 3.4 Основные виды дисторсии изображений: а – радиальная, б - тангенциальная

Таким образом, для устранения основных искажений необходимо определить пять корректировочных коэффициентов. В большинстве функций

OpenCV эти коэффициенты используются вместе, поэтому их объединяют в заданном порядке в матрицу дисторсии:

$$[k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3]$$

OpenCV для калибровки камеры предлагает функцию «calibrateCamera», которая вычисляет матрицу дисторсии. Метод калибровки заключается в том, чтобы направить камеру на известную структуру с большим количеством легко идентифицируемых точек. Съемка такого объекта под разными углами через вычисление относительного положения и ориентацию камеры позволяет определить внутренние параметры камеры. Из практических соображений для калибровки выбирают плоский объект с регулярным узором на поверхности, например, шахматную доску [186]. Функция «findChessboardCorners» позволяет обнаружить координаты углов на шахматной доске, которые необходимы для выполнения функции «calibrateCamera». Полученную матрицу дисторсии сохраняют в XML файл.

Этап калибровки камеры проводится один раз. В дальнейшем, при получении изображения с известной камеры, необходимо загрузить матрицу дисторсии из файла и передать ее в функцию «undistort», устраняющую искажения на изображении.

Подготовка чертежа. Для выполнения оцифровки, изображение должно содержать реперные маркеры – объекты, которые используются в качестве точки отсчета или меры. В нашем случае, реперные маркеры должны определять положение, границы и масштаб чертежа на изображении.

Также, как и в случае с калибровкой по шахматной доске, наиболее подходящей формой маркеров являются изображения на плоскости. В библиотеку OpenCV встроены алгоритмы поиска маркеров ArUco, зарекомендовавших себя в робототехнике и дополненной реальности. Каждый

маркер представляет собой чёрный квадрат с ярко выраженными границами, внутри которого располагается белый узор, однозначно его идентифицирующий. Однако, создание таких маркеров требует применения специального программного обеспечения, что может стать препятствием при самостоятельном обслуживании системы оцифровки предприятием. Поэтому нами предлагается использовать реперные маркеры в форме окружности, что позволит проводить подготовку чертежа к оцифровке при помощи подручных средств.

Для идентификации все маркеры необходимо выделить наименее встречающимся на изображении чертежа цветом. Фоном чертежа, как правило, являются светлые цвета, близкие к белому, а элементы чертежа выполняются в темных тонах, близких к черному. Соответственно, маркеры должны быть яркими и насыщенными, чтобы не сливаться с фоном и элементами чертежа. Яркость, насыщенность и оттенок являются компонентами цветовой модели HSV (Hue, Saturation, Value), что делает ее более подходящей для работы с изображением, чем стандартная модель RGB. В работах, изучающих цвет для технологии хромакея, совмещения нескольких изображений, выделены зеленый и синий цвет, ввиду их компонентного отсутствия в цвете кожи человека. При этом зеленый цвет поглощает меньше света, что делает его легким для обнаружения на изображении со слабым или неравномерным освещением. Таким образом, яркий и насыщенный зеленый цвет становится наиболее подходящим цветом для реперных маркеров. Определим это формально.

Допустим, R, G, B – красный, зеленый и синий компоненты пикселя, принимающие значения на интервале $[0,1]$. $H \in [0,360)$ – оттенок, $S \in [0,1]$ – насыщенность, и $V \in [0,1]$ – яркость пикселя. Тогда, преобразование пикселя из модели RGB в модель HSV выполняется следующим образом:

$$max = Max(R, G, B)$$

$$min = Min(R, G, B)$$

$$H = \begin{cases} 0 & \text{если } max = min \\ 60 * \dfrac{G - B}{max - min} & \text{если } max = R, \text{и } G \geq B \\ 60 * \dfrac{G - B}{max - min} + 360 & \text{если } max = R, \text{и } G < B \\ 60 * \dfrac{B - R}{max - min} + 120 & \text{если } max = G \\ 60 * \dfrac{R - G}{max - min} + 240 & \text{если } max = B \end{cases}$$

$$S = \begin{cases} 0 & \text{если}, max = 0 \\ 1 - \dfrac{min}{max} & \text{иначе} \end{cases}$$

$$V = max$$

Экспериментальным путем установлено, что яркий насыщенный зеленый цвет находится в следующих границах:

$$75 \leq H \leq 160 \; 0.5 \leq S \leq 1 \; 0.5 \leq V \leq 1$$

Рассмотрим расположение реперных маркеров. Область интереса представляет собой прямоугольник на плоскости, описывающий чертеж. Соответственно, ключевыми являются маркеры, размещенные на углах данного прямоугольника. Для приведения чертежа к реальным размерам при помощи маркеров необходимо зафиксировать их радиус или расстояние между ними. Вычисление радиуса маркера может быть затруднено плохой освещенностью – границы окружности будут размыты. В свою очередь, размеры чертежа могут быть разные, что делает невозможным установку постоянных размеров области интереса. Решением проблемы являются промежуточные реперные точки, лежащие на сторонах прямоугольника области интереса. Расстояние между точками задано и известно системе оцифровки. Соответственно, размеры

прямоугольника кратны величине расстояния между соседними реперными маркерами. Также, на изображении может присутствовать несколько прямоугольников, размеченных реперными точками. Предположим, что области интереса могут быть вложены друг в друга и иметь общие стороны. При обработке точек выбирается прямоугольник с самой большой площадью как основная область интереса (рис.3.5).

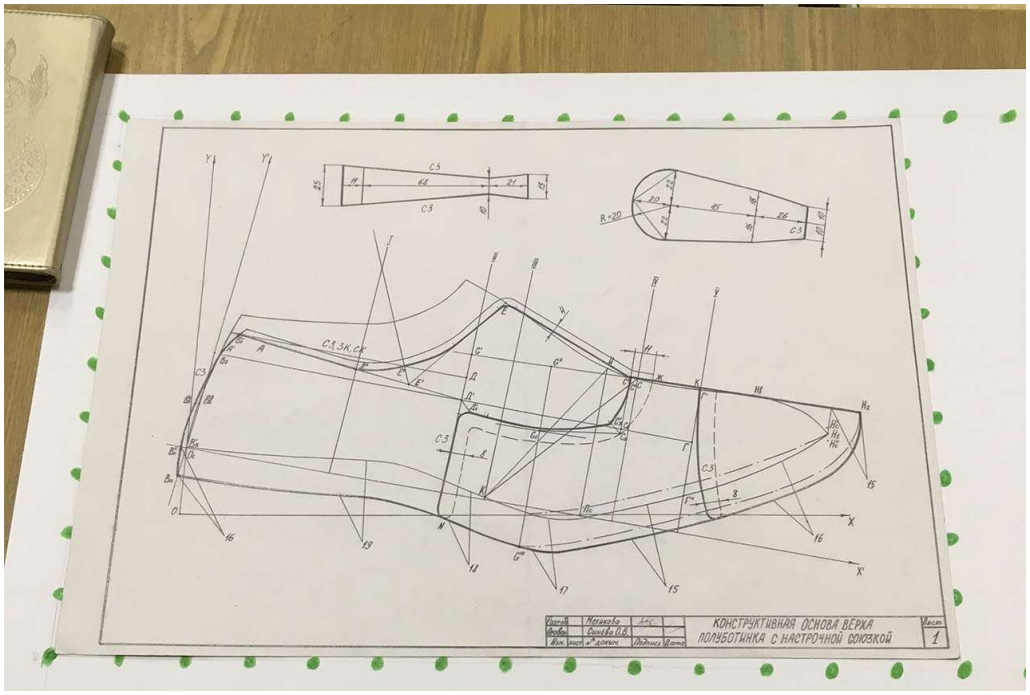


Рисунок 3.5 Размещение реперных точек на полотне под чертежом

Улучшение видимости элементов изображения. Перед выполнением поиска реперных маркеров на изображении, необходимо провести предварительную обработку изображения – скорректировать яркость и контраст, чтобы убрать тени и улучшить видимость чертежа [186]. Для этого, используя статистическую информацию частот различных уровней яркости [190], необходимо выровнять гистограммы изображения. Выравнивание сводится к такому преобразованию, после которого все уровни яркости пикселей изображения имеют примерно одинаковую частоту. При этом гистограмма будет иметь равномерный закон распределения [190].

В OpenCV выравнивание гистограммы выполняется функцией «equalizeHist». Однако, это базовый метод, который был многократно модифицирован для улучшения эффективности работы. OpenCV обладает набором функций для выполнения ограниченного адаптивного выравнивания гистограммы (CLAHE), который следует использовать при обработке чертежа. Изображение разбивается на множество блоков, для

каждого из которых выполняется выравнивание гистограммы – метод адаптивного выравнивания (AHE). В свою очередь, CLAHE вносит ограничение контрастности, что позволяет уменьшить количество шума [190]. Применив контрастно ограниченное адаптивное выравнивание гистограммы, шум может проявиться в высококонтрастных участках изображения. Учитывая, что изображение не было приведено к бинарной форме, такой шум будет хроматическим. Зеленые пиксели шума могут быть восприняты за реперные точки. В OpenCV реализован алгоритм очистки от шума FNLMD (Fast Non-Local Means Denoising). Этот алгоритм для усреднения цвета точки использует не соседние пиксели, а схожие пиксели во всем изображении [190]. При этом, определение схожести пикселей определяется не по цвету или яркости, а по окружению, в котором он находится. Основу алгоритма определяет идея о повторяющихся на изображении структурах. Поэтому даже если данный пиксель искажен шумом, то может найтись много похожих неискаженных пикселей. Поиск похожих пикселей производится с помощью квадрата $B(p,s)$ размера s с центром в пикселе p. Исходя из алгоритма [190], проводится сравнение данного квадрата с аналогичным ему, но в пикселе q: $B(q,s)$. Тогда, расстояние между $B(p,s)$ и $B(q,s)$ будет:

$$d^2\big(B(p,s),B(q,s)\big)=\frac{1}{3(2s+1)}\sum_{c=1}^{3}\sum_{j\in B(0,s)}\big(I_c(p+j)-I_c(q+j)\big)^2$$

Здесь c - индекс цветового канала, $I_c(p)$ – яркость канала c в точке p, а суммирование производится по всем пикселям в квадратной области. Пользуясь этим расстоянием, каждому пикселю можно назначить вес:

$$w(p,q) = e^{\frac{Max(d^2-2\sigma^2,0)}{h^2}},$$

где σ- ожидаемое стандартное отклонение шума в единицах яркости, а h – обобщенный параметр фильтрации, определяющий, насколько быстро убывает значимость квадрата по мере его удаления от корректируемой области. В общем случае чем больше h, тем больше шума будет удалено, но ценой некоторого снижения детальности изображения. Чем дальше (в пикселях) квадрат от обновляемого пикселя, тем меньше польза от него, потому что число таких квадратов с увеличением расстояния растет квадратично. Поэтому определяется некоторое окно поиска, и при обновлении учитывается только вклад квадратов в этой области [190]. Описанный алгоритм используется в функции «fastNlMeansDenoising», которую необходимо вызвать для подавления шума на изображении чертежа. Результат выравнивания гистограммы и устранения шума представлен на рисунке 3.6.

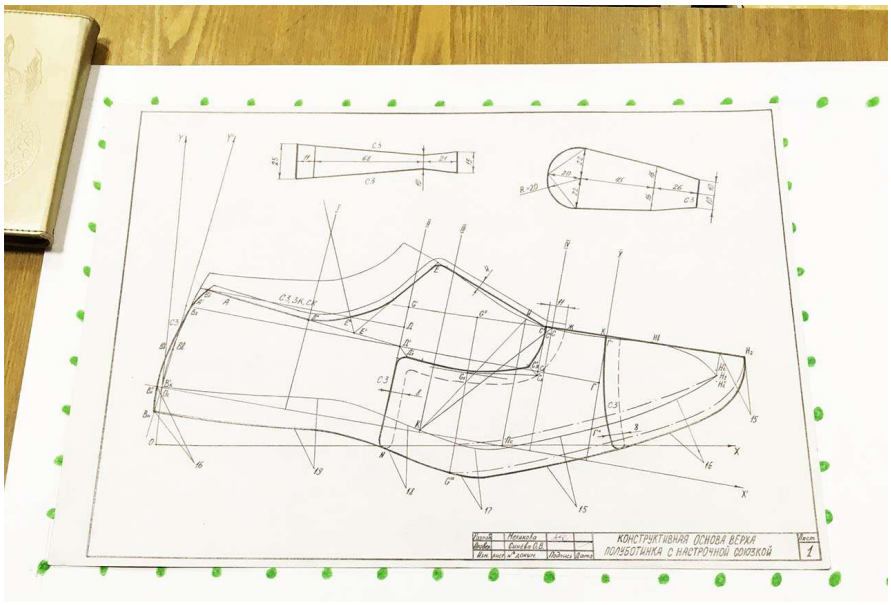


Рисунок 3.6 Изображение с выровненной гистограммой

Обнаружение реперных маркеров. Перед началом поиска, изображение следует конвертировать из цветовой модели RGB в HSV. Затем при помощи функции OpenCV «inRange» необходимо получить все яркие насыщенные зеленые пиксели, указав описанные раннее границы зеленого цвета в модели HSV. Результатом работы станет бинарное изображение, содержащее только пиксели реперных точек (рис. 3.7).

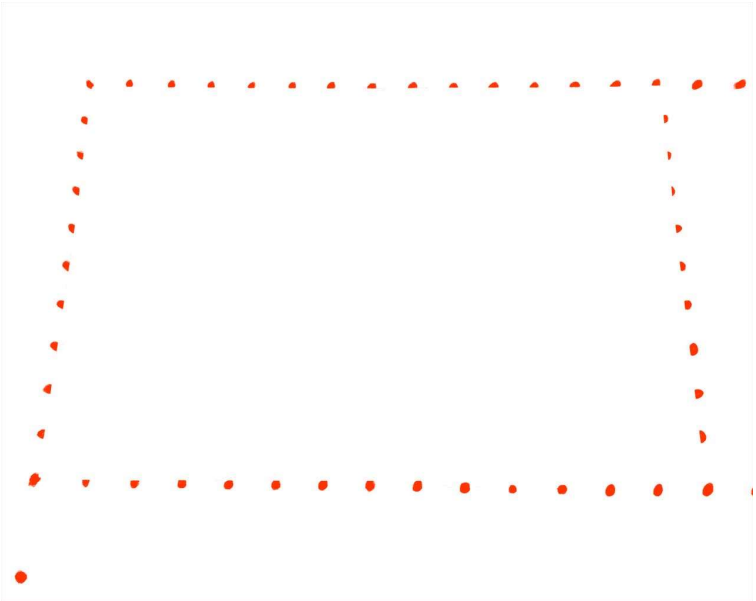


Рисунок 3.7 Бинарное изображение выделенных реперных точек

Такое изображение можно использовать, как аргумент функции «findContours», для поиска контуров (рис 3.8).

Несмотря на выравнивание гистограммы, отсутствие теней не гарантировано, и маркеры на бинарном изображении могут иметь разные формы и размеры. Поэтому координатами реперных точек будут являться центры найденных контуров. Для каждого контура i при помощи функции «moments» следует вычислить матрицу моментов M и затем получить центр следующим образом:

$$C_i = \left\{ \frac{M_{i,1,0}}{M_{i,0,0}}, \frac{M_{i,0,1}}{M_{i,0,0}} \right\}$$

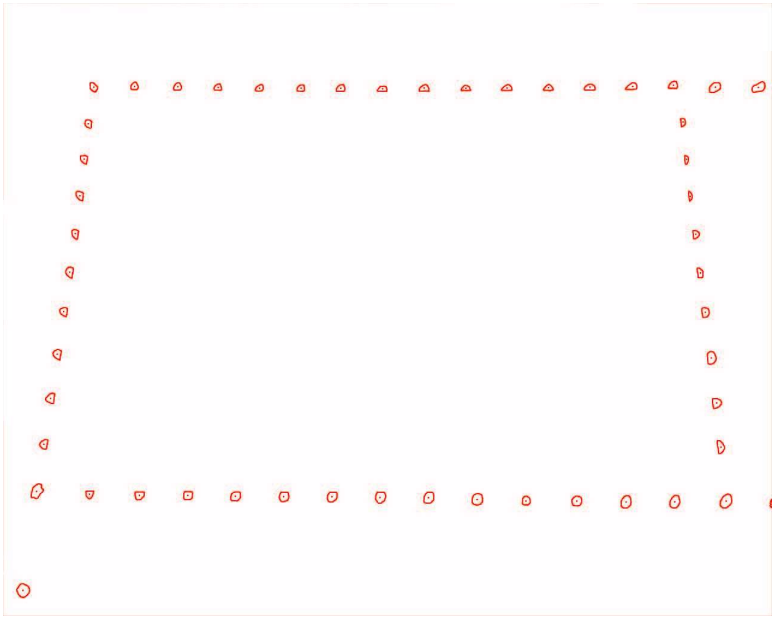


Рисунок 3.8 Результат работы алгоритма поиска контуров

С учетом того, что некоторый шум мог остаться после предварительной обработки, множество C может содержать ложные центры реперных точек. Так как изображение, как было отмечено ранее, может содержать множество непересекающихся прямоугольников реперных точек, разных размеров, задачей является поиск на изображении наибольшей области интереса.

Пусть на изображении задано n точек. Требуется найти цепочки точек, лежащих на прямых линиях. Одним из возможных решений является поиск всех прямых, определяемых каждой парой точек, с последующим формированием возможных цепочек, близких к конкретным прямым. Вычислительная сложность такого подхода не позволяет применить его в данной задаче. Альтернативным подходом является преобразование Хафа [190], в основу которого определяет следующее заключение. Если уравнение прямой на плоскости с угловым коэффициентом $y = ax + b$ записать в виде – b =– xa +

у и рассмотреть плоскость ab, называемую пространством параметров, то для заданной пары (x_i, y_i) можно получить уравнение единственной прямой. Другой точке (x_j, y_j) также соответствует своя прямая в пространстве параметров. Пересечение этих прямых в точке (a', b'), такой, что a' – это угловой коэффициент, b' – точка пересечения с осью у прямой, проходящей через точки (x_i, y_i) и (x_j, y_j) в плоскости xy. В OpenCV преобразование Хафа выполняется функцией «HoughLines» (рис. 3.9).

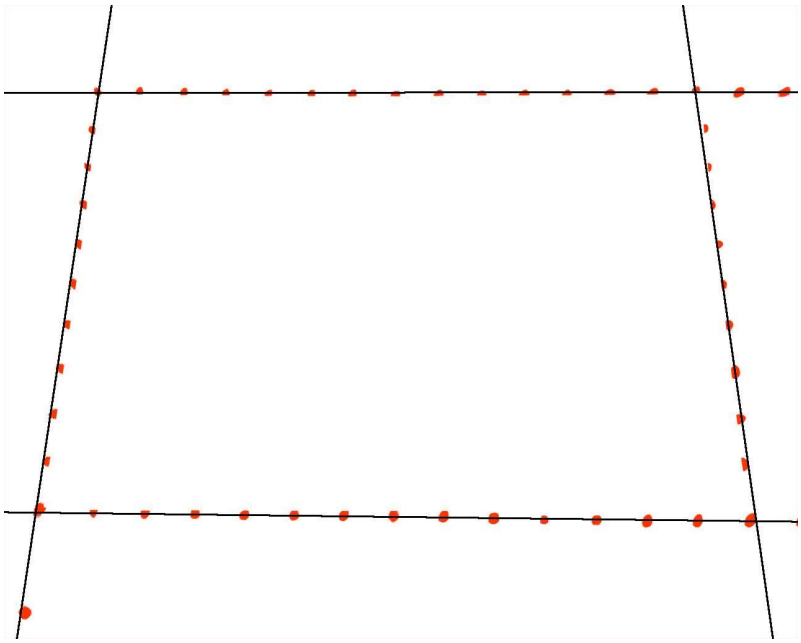


Рисунок 3.9 Визуализация линий, полученных преобразованием Хафа

Центры реперных точек необходимо проверить на принадлежность полученным линиям. Для этого следует создать пустое бинарное изображение с размерами оригинального изображения. Затем последовательно отрисовать каждую линию, проверяя заполненность пикселей с координатами реперных точек. Для каждой точки следует сохранить информацию об индексе линии, которой принадлежит точка. Точки, не принадлежащие линиям, следует удалить. Для каждой линии й должны существовать 2 угловые точки. Линии с меньшим числом угловых точек должны быть удалены. Это означает, что

информация о принадлежности точки к удаляемой линии должна быть очищена. Соответственно, после этой процедуры останется некоторое число точек без принадлежности к какой-либо линии, которые должны быть удалены из списка реперных точек.

После процесса фильтрации, список точек можно упорядочить. Первой станет точка, имеющая принадлежность нескольким линиям – это угол прямоугольника. Каждая следующая точка должна быть ближайшим соседом предыдущей и принадлежать одной линии.

Приведение к реальным размерам. На этом этапе необходимо устранить перспективные искажения и масштабировать изображение.

Для перспективного преобразования необходима матрица 3×3. Прямые линии после преобразования остаются прямыми, но пропорции изменяются [190]. Для поиска нужной матрицы, необходимо передать в функцию «getPerspectiveTransform» координаты четырех точек входного изображения и соответствующих им точек выходного изображения. В свою очередь, функция «warpPerspective» выполнит преобразование. В качестве точек входного изображения необходимо использовать угловые реперные точки. Для выходного изображения точки формируются из размеров области интереса. Рассмотрим подробнее порядок действий.

При обработке реперных точек нет возможности понять ориентацию изображения, например, при съемке под углом. Поэтому, первую угловую реперную точку следует выбрать случайным образом и принять ее за левый верхний угол. Затем, обойти все точки в случайном направлении до следующей угловой точки, и принять ее за правый верхний угол. Запомнить количество пройденных точек по верхней стороне, как CT, и продолжить обход по остальным углам. Также, запомнить количество точек по правой стороне CR,

нижней стороне CB и левой стороне CL. Количество точек на противоположных сторонах может отличаться, поэтому следует вычислить максимальное количество точек для длины и ширины:

$$MTB = Max(CT, CB)\ MLR = Max(CL, CR).$$

Расстояние между реперными точками LD заранее известно, тогда реальные размеры области интереса:

$$L = LD * MTB * \frac{DPI}{25.4}\ W = LD * MLR * \frac{DPI}{25.4},$$

где L, W реальная длина и ширина области интереса в пикселях. DPI – количество пикселей на дюйм, заданное исходным изображением. Отношение $\frac{DPI}{25.4}$ – перевод из дюймов в миллиметры. Тогда, в функцию «getPerspectiveTransform» для входного изображения передаются координаты углов в заданной последовательности: левого верхнего, правого верхнего, правого нижнего и левого нижнего. Для выходного изображения координаты прямоугольника: $\{0,0\}$, $\{W, 0\}$, $\{W, L\}$,$\{0, L\}$. Пример восстановления перспективы изображен на рисунке 3.10.

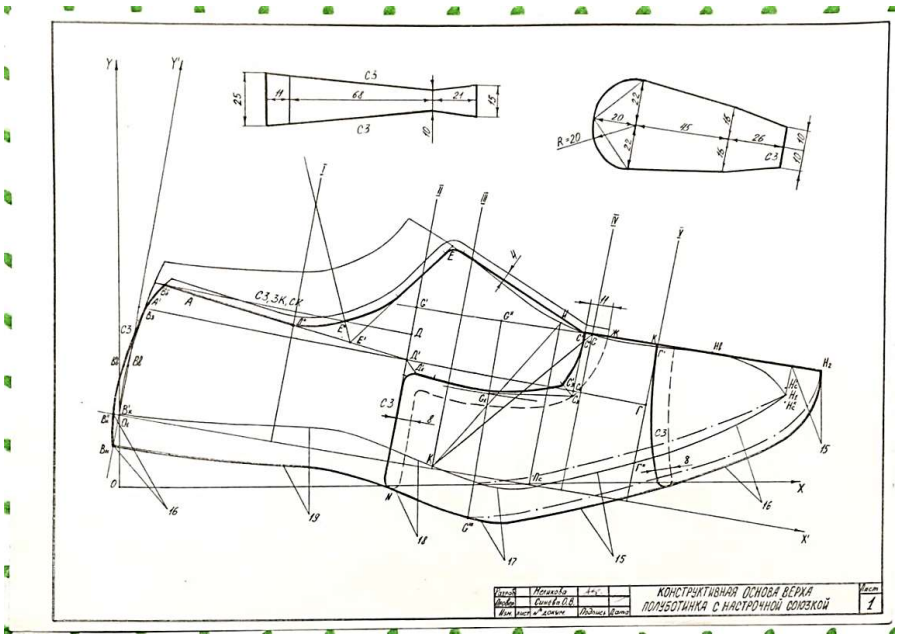


Рисунок 3.10 Результат восстановления перспективы

Устранение реперных маркеров. Финальный этап, задача которого закрасить реперные маркеры цветом фона, чтобы они не были приняты за элементы чертежа на этапе векторизации (рис. 3.11).

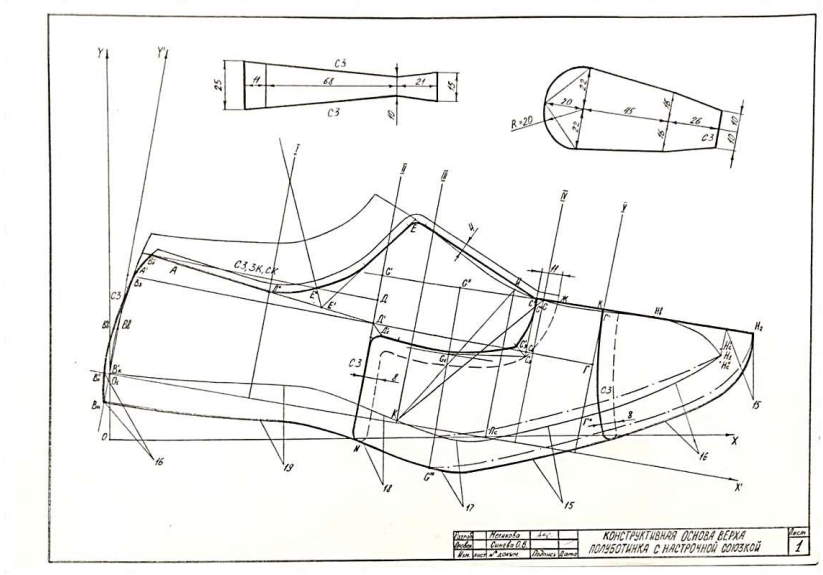


Рисунок 3.11 Изображение, подготовленное к процессу векторизации

Для этого необходимо выполнить перспективные преобразование над маской – бинарным изображением реперных точек, полученном раннее. В функцию «warpPerspective» передается та же матрица, что и при восстановлении перспективы оригинального изображения. В результате, изображение чертежа будет совместимо с маской.

После чего, закрашенные пиксели на маске можно закрасить на оригинальном изображении цветом фона, который в свою очередь можно получить из вычисленной ранее гистограммы. Качество проделанной операции не имеет значения, так как при векторизации к изображению будет применен пороговый фильтр.

Далее, полученное изображение векторизуют. OpenCV не обладает таким функционалом, однако есть множество других библиотек и программ, выполняющих векторизацию [186,191,192].

Лидером по качеству векторизации является программа с открытым исходным кодом Potrace, разрабатываемая с 2001 года и до сегодняшнего дня. Система использует алгоритм, который преобразовывает растровый рисунок в векторный контур за несколько этапов [193]:

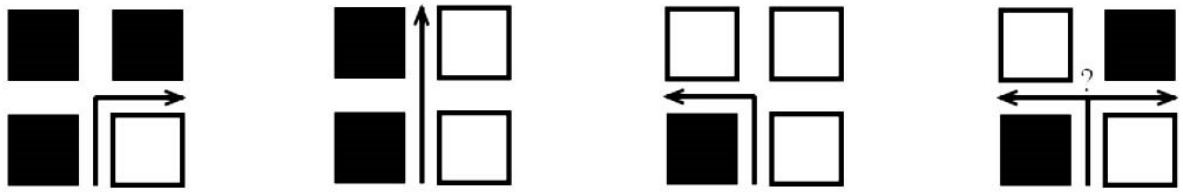
1. поиск возможных путей на изображении, которые формируют границы между черно-белыми областями;
2. формирование из каждого пути многоугольника;
3. каждый многоугольник интерполируется кривой;
4. опциональная оптимизация последовательных кривых.

Рассмотрим эти этапы подробнее.

Поиск пути. Для поиска контуров на изображении, выбирается пара смежных пикселей различного цвета. Это достигается, к примеру, выбором крайнего левого черного пикселя сверху. Соединение двух выбранных пикселей, формирующих край, осуществляется таким образом, чтобы черный пиксель был слева, а белый пиксель – справа. При прохождении по граням между пикселями и каждый раз при достижении угла либо выбирается прямое направление, либо осуществляется поворот между левым или правым пикселем, в зависимости от цвета окружающих пикселей, как показано на рисунке 3.12. Это происходит до тех пор, пока путь не дойдет до вершины, от которой он начинался, к точке, которая определяет закрытый путь [193,194].

В закрытый путь был найден, из графического контурного изображения удаляются и инвертируются все цвета пикселя в его внутренней области. Что формирует новое изображение, к которому применяется рекурсивный алгоритм к оставшимся там черным пикселям. В результате будет получен список закрытых путей, которые будут переданы к следующей стадии алгоритма. В ситуации, описанной на рисунке 3.12 (г), принимается решение о левом или

правом повороте. Выбор пути не влияет на результат работы алгоритма, поскольку любой путь будет замкнутым. Однако такой выбор оказывает влияние на описание путей. Ввиду этого, Potrace обладает параметром «turnpolicy», указываемый в командной строке [193]. Для обеспечения предсказуемости работы алгоритма следует выбрать фиксированный поворот налево или направо, вместо случайного выбора.



а б в г

Рисунок 3.12 Варианты обработки пикселей при поиске пути

Формирование многоугольника. Сначала алгоритм производит удаление пятен, путем разброса всех путей, внутренняя область которых имеет слишком малое количество пикселей. Затем, расставляет точки многоугольника в каждом черном пикселе (рис. 3.13), точно описывая таким образом путь [193].

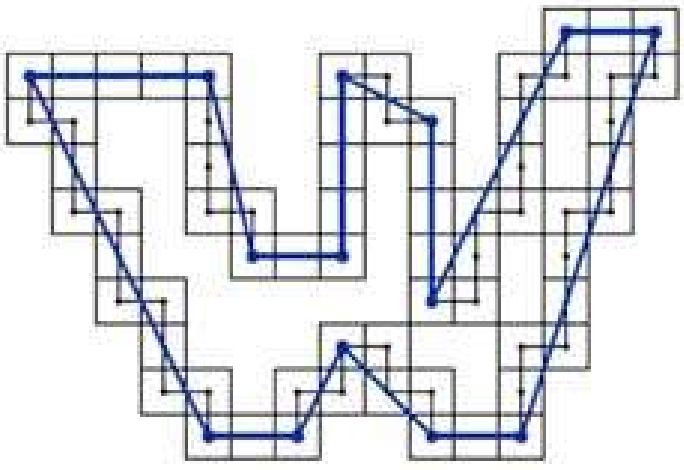


Рисунок 3.13 Результат формирования многоугольника

Интерполяция. Алгоритм производит преобразование многоугольника, полученного на предыдущей стадии, в сглаженный векторный контур. Предварительно производится коррекция позиции вершин многоугольника для лучшего соответствия первоначальному рисунку, насколько это возможно. Затем определяются углы и кривые, основанные на длинах смежных сегментов линии и углах между ними. Результатом анализа углов и сглаживания станет кривая, состоящая из сегментов кривой Безье и сегментов прямой линии [193].

Оптимизация. Для улучшения кривой производится соединение смежных сегментов кривой Безье (рис. 3.14).

Оптимизация кривой приводит к незначительным е изменениям форм конечной кривой, и они настолько малы, что различий обычно не видно. Однако окончательная кривая состоит из меньшего количества сегментов и, таким образом, может быть представлена более сжато в финальной стадии программы [193].

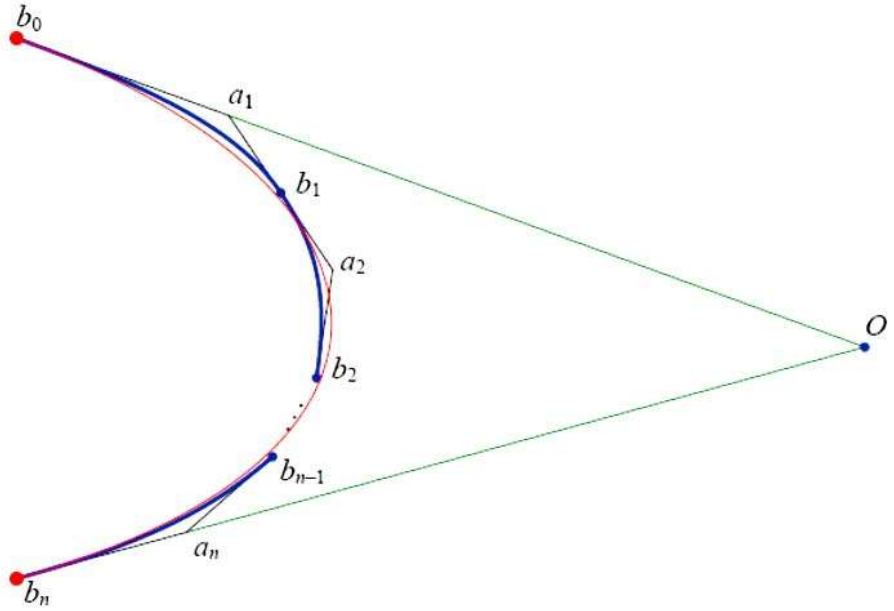


Рисунок 3.14 Оптимизация кривой

Перейдем к описанию сервисов. Так как калибровку камеры достаточно проводить один раз, в облачной САПР в процессе оцифровки будут задействованы 2 сервиса: сервис управления камерами и сервис оцифровки.

Сервис управления камерами – должен хранить информацию о калибровке пользовательских камер и проводить процесс калибровки. Основываясь на описанном процессе калибровки, сообщениями, которые должен обрабатывать сервис управления камерами, будут:

- CMCreateCam принимает на вход имя камеры и создает запись о камере в базе данных. Такой записи присваивается уникальный идентификатор, сохраняется переданное имя, создается пустое XML содержимое настроек калибровки, а также поле с информацией об обработке. Сообщение возвращает идентификатор новой записи и обрабатывается синхронно, так как не требует большого количества вычислительных ресурсов.
- CMDeleteCam принимает на вход идентификатор камеры и удаляет запись о ней из базы данных. Так же, как и создание записи, сообщение обрабатывается синхронно.
- CMGetCam принимает идентификатор камеры и возвращает запись о ней из базы данных по ее идентификатору, включая XML содержимое настроек калибровки и информацию об обработке. Сообщение обрабатывается синхронно.
- CMGetCamList возвращает список названий и идентификаторов камер в базе данных. Содержимое настроек калибровки не возвращается, вместо этого в ответ добавляется отметка заполнено ли содержимое настроек в базе данных (была ли проведена калибровка). Сообщение обрабатывается синхронно.
- CMCalibrateCam принимает идентификатор камеры и изображение шахматной доски. По описанному раннее алгоритму проводится

калибровка, и настройки сохраняются в базе данных. Информация об ошибке обработки помещается в соответствующее поле записи базы данных. Сообщение обрабатывается асинхронно, и в результате порождает сообщение CMCamCalibrated. включающее идентификатор камеры, с которой проводилась работа, или ошибку обработки.

- CMIsCalibratedCam принимает идентификатор камеры и возвращает из базы данных информацию об обработке вместе с флагом заполненности настроек калибровки. Сообщение обрабатывается синхронно.

Процесс работы пользователя с сервисом через веб-приложения проходит в следующем порядке. Приложение отправляет в систему сообщение CMCreateCam с именем новой камеры для создания записи в базе данных и получает в ответ идентификатор новой камеры. После чего приложение отправляет в систему сообщение CMCalibrateCam с идентификатором камеры и файлом шахматной доски. Так как сообщение обрабатывается асинхронно, приложению следует периодически отправлять в систему сообщение CMIsCalibratedCam с идентификатором камеры. В ответ приходит информация о калибровке камеры или ошибке обработки изображения.

Процесс межсервисного взаимодействия идентичен, за исключением опроса системы сообщением CMIsCalibratedCam. Сервису достаточно подписаться на сообщение CMCamCalibrated, по окончании процесса калибровки, чтобы узнать результат обработки.

Основываясь на описанном процессе предварительной обработки и векторизации, выделим сообщения, которые должен обрабатывать сервис оцифровки:

- IVProcess должен быть передан вместе с изображением и идентификатором камеры в базе данных. Сервис сохраняет изображение в облачном

хранилище, создает запись в базе данных о загруженном файле. Запускается процесс оцифровки и векторизации. В базе данных создается запись о начале процесса. Результат сохраняется в облачном хранилище. Сообщение обрабатывается асинхронно. По окончанию работы порождает сообщение IVProcessed, содержащий идентификатор записи обработки изображения из базы данных.

- IVClean, преданный с идентификатором записи обработки, сигнализирует о том, что данную запись нужно удалить, вместе с файлами из облачного хранилища. Обрабатывается синхронно.
- IVGetList возвращает список идентификаторов файлов, находящихся на обработке в сервисе оцифровки. Обрабатывается синхронно.
- IVGetImage возвращает изображение по переданному идентификатору. Обрабатывается синхронно.
- IVGetVector возвращает результат векторизации по переданному идентификатору. Обрабатывается синхронно.
- IVGetSettings возвращает набор настроек обработки изображения из базы данных по переданному идентификатору. Обрабатывается синхронно.
- IVGetStatus возвращает информацию, находится ли запись в обработке. Обрабатывается синхронно.

Работа пользователя с сервисом оцифровки через веб-приложение проводится в следующем порядке. Сначала пользователь выбирает камеру, которой была выполнена съемка изображения при помощи CMGetCamList. Затем загружает изображение чертежа с идентификатором камеры, отправляя сообщение IVProcess, запуская процесс оцифровки. Во время ожидания периодически отправляется запрос IVGetStatus, который покажет закончена ли обработка.

Результатом работы программы векторизации является описание графических элементов в некотором стандартизированном общем формате. Использовать такой формат для хранения проектных данных не всегда выгодно, так как он описывает только графику, без семантики. Результат векторизации должен быть конвертирован другим сервисом, подписанным на сообщение IVVectorized, в общий формат проектных данных. Рассмотрим формат, который можно использовать для проектных данных.

3.3 Формирование модели проектных данных

Результатом работы Potrace, как и многих других программ векторизации, является файл SVG (Scalable Vector Graphics) – это язык разметки векторной графики, созданный Консорциумом Всемирной паутины (XSAZ) в 1999 году [195]. Системы векторизации используют не весь стандарт для описания результата. Зачастую, применяется только один элемент.

Path – это самый универсальный и гибкий элемент SVG для отрисовки прямых, кривых и фигурных линий, а также сложных открытых и замкнутых контуров. Программы векторизации чаще всего используют именно этот элемент. Path имеет атрибут «d», внутри которого находится пакет команд, определяющих траекторию и направление фигурной линии. Список возможных команд: перемещение курсора, линия, горизонтальная линия, вертикальная линия, дуга эллипса, квадратичная кривая Безье, кубическая кривая Безье, замыкание линии [195]. Такой формат очень удобен для применения в веб-приложении, но документы в таком формате очень громоздкие, и их сложно хранить в базе данных [196].

Для уменьшения объема занимаемого места, конструкцию верха обуви следует описывать одним геометрическим примитивом. В качестве такого

примитива нами выбран интерполяционный сплайн Catmull-Rom. Проведенные исследования показали, что интерполяционный сплайн Эрмита, и его частный случай Catmull-Rom подходят для описания конструкции верха обуви [191,197]. Данный сплайн очень компактный, так как позволяет хранить в базе данных только ключевые точки. Также, особенность сплайна Catmull-Rom заключается в том, что он может быть преобразован в набор кубических кривых Безье [198].

Кривые Безье являются одним из основных элементов векторной графики. Независимо друг от друга кривые Безье были разработаны двумя инженерами из французских автомобильных компаний Полем де Кастелье («Ситроен») и Пьером Безье («Рено») для проектирования кузовов автомобилей. Несмотря на то, что де Кастелье изобрел их раньше, первым достижение представил публике Пьер Безье в 1962 году. В последствии кривые Безье стали применяться не только в автомобилестроении, но и в системах автоматизированного проектирования и программах компьютерной графики [199]. Учитывая требовательность к вычислениям, наибольшее распространение получили кубические кривые Безье. Так как последовательно соединенные кубические кривые могут описать любую геометрическую форму, и не требовательны к вычислительным мощностям. Сегодня кубические кривые Безье используются в технологиях отрисовки графики на системном уровне: GDI, X11, Qt (GNOME), GTK (KDE), OpenGL, Vulkan, Metal, DirectX. В крупных фреймворках векторной графики Skia и Cairo, а также в использующих их движках браузеров: Blink, WebKit, Gecko, Trident. Кривые Безье входят в стандарты SVG, HTML5, и CSS, на которых строятся веб-страницы. Задача отрисовки кривых Безье полностью ложится на браузер, что обеспечивает высокую скорость отрисовки кривых [78,200].

Таким образом, кривые Безье являются точкой соприкосновения проектных данных, использующих сплайн Catmull-Rom в качестве примитива, браузера и стандарта SVG, который оперирует системами векторизации.

Параметрическое представление кубических кривых Безье описывается следующим образом [199]:

$$P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3,$$

где $t \in [0,1]$. P_0 и P_3 – точки начала и конца кривой. P_1 и P_2 – управляющие точки, задающие кривизну. Для преобразования сплайна Catmull-Rom в набор кривых Безье и обратно выполняются следующие вычисления [198]:

$$\begin{matrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{matrix}_{Bezier} \Rightarrow \begin{matrix} P_4' + 6(P_1' - P_2') \\ P_1' \\ P_4' \\ P_1' + 6(P_4' - P_3') \end{matrix}_{Catmull-Ro}$$

$$\begin{matrix} P'_1 \\ P'_2 \\ P'_3 \\ P'_4 \end{matrix}_{Catmull-Rom} \Rightarrow \begin{matrix} P_2 \\ P_2 + \frac{1}{6}(P_3 - P_1) \\ P_3 - \frac{1}{6}(P_4 - P_2) \\ P_3 \end{matrix}_{Bezier}$$

где P_0', P_1', P_2', P_3' – это 4 последовательные точки сплайна Catmull-Rom.

В процессе разработки конструкции верха обуви выполняется построение множества элементов, имеющих разную семантику. Рассмотрим их с точки зрения графического представления [201]:

- базовая линия – произвольный сплайн;
- ось – сплайн, ограниченный двумя точками;
- припуск – линия, смещенная от базовой на фиксированное расстояние;

- переменный припуск – линия, смещенная от базовой на переменное расстояние.
- накол – графический объект, располагающийся на линии, смещенной от базовой;
- перфорация – массив графических объектов, располагающийся на линии, смещенной от базовой;
- гофра – графический объект, располагающийся на линии, смещенной от базовой;
- трафаретная линия – графический объект произвольного размера, располагающийся как на базовой линии, так и на смещении от базовой;
- точка – произвольно расположенный графический объект.

Из списка видно, что в модель данных необходимо ввести еще два примитива. Графический объект – это объект, с заранее заданной формой. Чтобы обеспечить изменяемость и динамичность этого объекта, для описания графических объектов подойдет система команд элемента Path из языка разметки SVG. Однако, чтобы обеспечить совместимость с основным примитивом, из всех команд для описания следует использовать только прямые и кубические кривые Безье. Помимо графического объекта, следует ввести примитив – зависимый объект. Это объект, форма или поведение которого зависят от другого объекта.

Сплайн и графический объект описывают только графические данные. Введение других элементов в модель приведет к появлению семантически значимых объектов. Таким образом, формируется иерархическая модель данных, основанная на уровнях абстракции [201]. Низший уровень абстракции предполагает отсутствие какой-либо семантики – это примитивы. Наивысший уровень абстракции предполагает наличие семантики и отсутствие какого-либо графического описания.

Так, следующим уровнем после примитива станут: базовая линия, ось, припуск, накол, гофра, трафаретная линия, точка. Затем, уровнем выше: переменный припуск, перфорация. На рис. 3.15 приведена диаграмма классов проектных данных.

Группировка описанных объектов производится при помощи классов: деталь обуви (объединение построенных объектов), проект (объединение всех объектов и деталей).

Различные сервисы моделирования могут работать с собственными моделями данных, но для обмена информацией между сервисами необходимо использовать унифицированную модель, описанную выше. Базовая функциональность вычислительного сервиса облачной системы автоматизированного проектирования обуви, помимо создания и удаления объектов, должна включать операции изменения:

- точек базовой линии и оси;
- указателя на базовый объект для постоянного и переменного припуска, накола, перфорации, гофры, трафаретной линии;
- положения точки, гофры, накола;
- значения и ориентации отступа постоянного и переменного припуска, трафаретной линии, перфорации;
- размера накола, перфорации, гофры, трафаретной линии, точки;

Разработанная модель проектных данных является оптимальной для облачной САПР обуви за счет минимизации объема хранимых данных и возможности конвертирования данной модели в открытые форматы. Однако при этом возникает вопрос, как представить такую объектную иерархию в базе данных. Перейдем к выбору системы управления базами данных, возможностей которых было бы достаточно для хранения не только представленных

проектных данных, но и данных различных сервисов, описанных при разработке ядра и оцифровки.

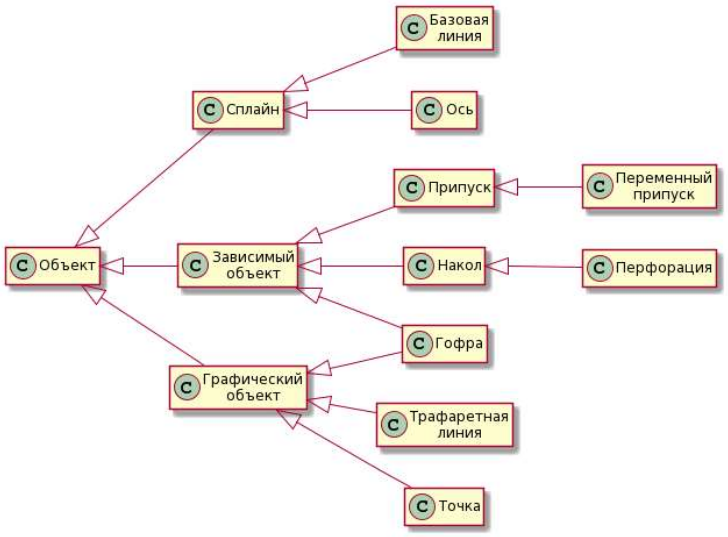


Рисунок 3.15 Диаграмма классов проектных данных

3.4 Выбор системы управления базами данных для хранения проектных данных

На сегодняшний день существует множество СУБД, которые можно разделить на 2 класса: реляционные и нереляционные.

Реляционная модель получила большое распространение для хранения структурированных данных клиент-серверных систем. Основными преимуществами реляционных баз данных является наличие:

1. математического реляционного описания;
2. методов оптимизации для поддержки параллельного выполнения задач в многопоточных системах;
3. Тьюринг-полного языка манипуляции данными SQL, который легок в изучении;

4. большой экосистемы, включающей как системы управления базами данных, так и инструменты проектирования структур данных.
5. механизма ACID транзакций [144] (атомарность, согласованность, изолированность, долговечность) и блокировок записей;

Недостатки реляционных баз данных проявляются с ростом объема обрабатываемых данных.

Реляционные базы данных не позволяют хранить объекты в явном виде. Например, чтобы отобразить всю информацию о модели обуви программист должен выполнить несколько запросов для объединения данных из многих связанных между собой таблиц: детали, материалы и т.д. При значительном росте числа таблиц осложняется связывание таблиц при выполнении запроса, разработчик можно просто забыть назначение той или иной таблицы, что затрудняет формирование объекта. Для устранения этих проблем были созданы инструменты объектно-реляционного отображения, однако они не смоги полностью устранить проблему [203].

Большие объемы информации необходимо хранить при помощи кластеров параллельных систем баз данных. При масштабировании таких систем возникают проблемы поддержания отказоустойчивости [204]. Сетевое взаимодействие кластеров, при выполнении сложных запросов, оказывает негативное влияние на производительность системы [205]. Элементы предметной области отражены в иерархичных структурах базы данных. Для модификация вложенной структуры и реорганизации таблиц необходимо остановить работу остальных таблиц. В качестве альтернативы, на сегодняшний день получили широкое применение «базы данных приложений», предполагающие сбор данных в единый объект на уровне прикладной программы, а не базы данных. При этом, задачи обеспечения целостности

данных и поддержания высокой производительности такого взаимодействия должны решаться разработчиками программ [205].

В распределенных реляционных базах данных операция репликации изначально выполнялась с использованием протокола «read-one-write-all» [206]: процедура чтения блокирует данные локально на узле, в то время как для операций записи происходит блокировка всех копий записи. Метод двухфазной фиксации в конце каждой транзакции гарантирует атомарность операции, таким образом каждая операция координируется отдельно [207]. В результате при увеличении числа реплик вероятность возникновения конфликтов и время реакции системы увеличиваются по экспоненте [208].

Рассматривая современные тенденции в области обработки данных, исследователи делают отмечают, что многие реляционные СУБД были разработаны более 25 лет назад, когда требования пользователей, объемы обрабатываемых данных, вычислительные мощности, и рынки баз данных отличались от сегодняшних. Способы применения баз данных развивались с 1970-х годов, когда была только обработка бизнес-данных. Пользовательские интерфейсы и модель использования также изменились за последние несколько десятилетий [209].

Сегодня активно развиваются «нереляционные базы данных» (NoSQL), как результат попыток решить накопившиеся проблемы реляционных баз данных. Термин NoSQL применяется, чтобы описать большой класс баз данных, которые имеют свойства отличные от реляционных базы данных и уникальный язык запросов. Этот термин был введен компаниями Google и Amazon, которые используют технологию нереляционных систем для хранения обработки больших объемов данных [210,211]. Рассмотрим преимущества нереляционных баз данных.

Масштабируемость. Нереляционные методы хранения данных очень хорошо поддерживают горизонтальную масштабируемость – разделение данных на множество кластеров. А также позволяют обрабатывать большое количество простых задач в единицу времени.

Параллельные вычисления. Многие нереляционные базы данных имеют в своем составе средства поддерживающие параллельные вычисления [212].

Простота проектирования модели данных. Данные хранятся без четкой структуры. Разные записи могут иметь разную структуру, поддерживая вложенность объектов. В полях объектов можно сохранить другие объекты. Таким образом нет необходимости читать данные из разных таблиц, а при разработке нет необходимости разрабатывать структуру базы данных, как это принято в реляционных базах данных.

Открытость. Большинство нереляционных базы данных распространяются под свободными лицензиями с открытым исходным кодом. Это делает их бесплатными и позволяет работать на разных операционных системах.

Современные интерфейсы взаимодействия. Многие нереляционные системы выполняют операции получения или обновления записи посредством сетевые HTTP запрос, в формате стандарта REST [213,214].

Однако, нереляционные базы данных обладают множеством недостатков, возникших из-за отказа от ограничений ACID для обеспечения масштабируемости и высокой производительности.

Негарантированная надежность. Нереляционные базы данных невозможно использовать в приложениях, где необходимы: атомарность, согласованность, изолированность и стойкость данных (ACID) [215].

Частичная согласованность. Строгая согласованность данных приводит к увеличению времени выполнения операций. Ввиду этого, согласованность копий записей на разных кластерах обеспечивается с некоторой задержкой, из-за распространения информации об обновлении на все реплики хранилища данных. Возможны ситуации чтения устаревших данных.

Сложность обработки данных. Сложность программирования запросов к нереляционным базам данных возрастает, так как для доступа к данным применяются процедурные языки. В результате, для реализации сбора данных об объекте выполняется несколько операций, подобно «базам данных приложений».

Производительность. Отказ от поддержки ACID-транзакций оказывает положительное влияние на производительность нереляционных баз данных. Однако, некоторые СУБД, например Riak, стараются сохранить некоторые свойства ACID [216]. Так, реализация возможности выполнения определенных процедур до и после фиксации данных – функционирует, как триггеры реляционной базы данных [217]. Агрегация всех изменяемых данных в одной записи базы данных позволяет сохранить свойство атомарности [203]. Однако, при необходимости обновления нескольких связанных объектов, атомарность не гарантируется. Сохранить свойство изолированности в нереляционных базах данных невозможно, из-за отсутствия функций блокировки обновляемых записей.

В нереляционных системах данные хранятся в виде записей «ключ-значение», при этом значения могут иметь различные типы данных, на основе которых формируются способы функционирования базы данных. Выделяют следующие типы данных: «ключ-значение», документ, «семейство столбцов», графы [218,219]. Такой набор видов данных делает системы объектно-ориентированными [203]. Наиболее распространенный тип данных «ключ-

значение» является неявным для хранилища данных. Так как структура значения динамична, система управления базами данных не может получить информацию о каком-либо вложенном объекте без получения полной записи. Таким образом, структуру объекта видит только приложение, а чтение записи выполняется по ключу. Для простых атрибутов, помещенных в заголовок записи, возможен поиск по индексу. Примерами баз данных «ключ-значение» являются Riak, Redis, Amason Dynamo.

Остальные типы данных значений являются явным и системы управлениями базами данных способны получить информацию о структуре объекта. Это позволяет реализовать чтение и запись не всего объекта, а только его полей. Помимо этого, возможна реализация полноценного индексирования полей объектов для повышения производительности операции поиска.

В документной базе данных каждый объект обладает уникальным идентификатором, используемым в запросах, как ключ [220]. Объект-документ представляет собой некоторый блок информации, в определенном базой данных формате. Например, объекты в базе данных MongoDB хранятся в формате JSON, рассмотренном ранее. Для оптимизации объема хранимых данных возможно применение формата бинарного JSON – BSON (Binary JavaScript Object Notation) [221]. Примерами документных баз данных являются MongoBD, CouchBD.

Базы данных, использующих «семейство столбцов» в качестве типа данных объектов, были одними из первых, на пути к формированию класса нерялиционных баз данных. Поэтому их работа во многом похожа на подходы в традиционных реляционных базах данных. Наборы столбцов в разных записях могут быть разными. Примерами этого типа баз данных являются BigTable, HBase, Cassandra.

В графовой базе данных объекты представляются в виде наборов вершин и связей между ними. Такой подход ориентирован на быструю навигацию между записями. Каждая запись в базе данных – это пара «вершина-связь», что позволяет при чтении одного свойства перейти к другому свойству. Примером такой базы данных является Neo4j.

Наиболее удобной для разработки и хранения проектных данных, рассмотренных в разделе 3.3, являются нереляционные документные СУБД. Каждый проект может быть представлен документом. В свою очередь, отсутствие схемы базы данных позволяет реализовать классовое наследование. Однако при этом данные, рассмотренные в разделах 3.1 и 3.2 – реляционные. Например, для информации о регистрации ядра, сервисов, ресурсов лучше подходят реляционные модели баз данных.

Для решения этой проблемы небольшое количество активно разрабатываемых реляционных СУБД вводят нереляционную функциональность. Одной из таких систем является PostgreSQL – объектно-реляционная система управления базами данных с открытым исходным кодом [222]. Данная СУБД конкурирует наравне с коммерческими базами данных: Oracle, SQL Server и другими [223]. PostgreSQL развивалась как исследовательский проект в Калифорнийском университете Беркли. С 1996 года она разрабатывалась сообществом и до сих пор в разработке PostgreSQL принимают активное участие сообщества и университеты. Сегодня PostgreSQL используется во многих отраслях, например телекоммуникации, медицина, географические приложения и электронной коммерции. Система PostgreSQL набирает популярность в госструктурах, проводящих импортозамещение инфраструктурных компонентов систем [224]. Система управления базами данных PostgreSQL обладает возможностью хранить данные в формате JSON [225]. Для этого в системе реализовано два типа: JSON и JSONB, тот и другой

реализуют спецификацию RFC 7159. Оба типа применимы для контроля правил JSON. Они почти идентичны, но JSONB эффективнее, потому что является двоичным форматом и поддерживает индексы. JSON-объекты могут содержать вложенные JSON-объекты, массивы, вложенные массивы, массивы JSON-объектов и т. д. Глубина вложенности произвольна, так что можно конструировать весьма сложные JSON-документы. В одном массиве JSON-документа могут находиться элементы разных типов.

Проведенное сравнение производительности PostgreSQL с нереляционными системами Apache Cassandra и Amazon DynamoDB [226] показало, что PostgreSQL выполняет простые операции чтения, вставки и обновления записи не хуже, а в некоторых случаях лучше, исследуемых нереляционных систем. Операция чтения из соединенных таблиц выполняется в реляционной базе данных медленнее за счет объединения таблиц внутри нереляционных баз данных и выполнения для них простой операции чтения. Такое решение, в свою очередь, влечет за собой увеличение объема хранимых данных за счет избыточности при значительном различии ссылающейся и ссылаемой таблиц. Операция подсчета сгруппированных записей выполняется PostgreSQL быстрее, так как при работе с нереляционными базами данных основные вычисления оставались на стороне пользователя [226].

Таким образом, объектно-реляционные системы управления базами данных, подобные PostgreSQL, позволят обеспечить высокую производительность, доступность и надежность для всех видов данных облачной САПР обуви.

Разработанная нами архитектура и предложенные компоненты упрощают внесение фундаментальных изменений в САПР, что обеспечивает ее эволюционное развитие. Рассмотрим направления развития концепции облачной САПР обуви.

3.5 Направления развития концепции облачной САПР обуви

Разработку облачной САПР обуви можно разделить на 2 этапа. **Подготовительный этап** разработки заключается в построении «минимального жизнеспособного продукта» (minimum viable product), и состоит из нескольких шагов.

1. Анализ условий среды выполнения САПР обуви: развертывание в частном или публичном облаке, географическое расположение вычислительных узлов, возможная нагрузка.
2. Выбор начальной формы архитектуры. Производится исходя из анализа условий среды.
3. Разработка отладочного ядра. Функции отладки позволяют не только избавиться от возможных ошибок, но и ускоряют процесс разработки.
4. Развертывание базы данных для разработки и тестов.
5. Разработка рабочего ядра. Производится путем сокращения отладочного функционала, разработанного раннее отладочного ядра.
6. Разработка минимального функционала САПР обуви, а именно сервисы: управления проектами, 2D конструирования, оцифровки, градирования, раскроя, экспорта.

Созданная в результате система будет представлять собой базу для облачной САПР обуви. Следующий **этап разработки – основной**, заключается в развитии созданной базы, для формирования конкурентоспособной САПР обуви. Можно выделить несколько направлений дальнейшего развития системы [227]. Совершенствование проектирования – это главное направление развития для облачной САПР обуви. К этому направлению, в первую очередь, относятся возможности автоматизации конструирования обуви в САПР, описанные в разделе 1.2. Помимо этого, актуальной является задача автоматизации

технологической подготовки: разработка технологических карт сборки заготовки обуви, а также генерация программ ЧПУ для разработки оснастки производства, автоматического разруба и раскроя материалов, различных автоматов, используемых на линиях производства обуви.

Ритейл-направление – это создание связи между облачной САПР обуви и конечным потребителем. Так, в настоящее время актуальна проблема кастомизации, как вариант адаптации массового продукта под потребности конкретного потребителя. Многие зарубежные производители обуви на своих официальных сайтах предоставляют такую возможность для своих покупателей при помощи онлайн-конструктора: обуви можно придать отличительные черты, изменив цвет материалов верха обуви, отделку и элементы фурнитуры. Результат работы онлайн-конструктора может быть интегрирован в облачную САПР обуви. Также развитие 3D-сканеров и рост их доступности позволили многим компаниям создать системы виртуального подбора обуви по данным 3D-сканирования стоп.

Технологии дополненной реальности позволяют визуализировать виртуальную примерку обуви. Так, в конце 2018 года Gucci представила новую технологию на основе AR, которая позволяет покупателям виртуально примерить кроссовки Ace [228]. Эта услуга, разработанная в партнерстве с белорусской технологической компанией Wannaby, доступна через мобильное приложение Gucci для iOS. В последнее время итальянский модный дом внедряет ряд ориентированных на потребителя инструментов дополненной реальности. В рамках услуги «Сделай сам», посвященной кроссовкам Ace, сумке Ophidia и избранным трикотажным изделиям, верхней женской одежде и индивидуальным деталям, Gucci оснастила ряд своих магазинов инструментом, позволяющим покупателям настроив iPad или камеру iPhone на конкретный продукт, доступный в магазине, просмотреть его в реальных условиях. В 2019

году интернет-магазин Lamoda также запустил виртуальную примерочную на базе вышеупомянутого белорусского AR-стартапа Wannaby [228]. Для виртуальной примерки доступно около ста самых популярных женских и мужских моделей – Nike, Adidas, Reebok, Converse, Under Armour и других марок самой востребованной категории – кроссовок. Виртуальная примерочная доступна только пользователям iPhone – начиная с модели 6S. Клиенты Lamoda могут примерить заказанную одежду и обувь перед тем, как принять окончательное решение о покупке. Формально функцию виртуальной примерки запустили и в интернет-магазине обуви компании Ecco. Онлайн-ритейлер Wildberries также использует приложение для 3D-моделирования стопы [228].

Непрерывный сбор статистики о действиях пользователей интернет-магазинов и ее дальнейший анализ позволяют компаниям лучше понять предпочтения покупателей. Интеграция этих процессов с облачной САПР обуви позволит до начала производства определить успешность разработанной коллекции обуви или отдельных конструктивных решений.

Корпоративное направление – это интеграция облачной САПР обуви с другими системами автоматизации бизнес-процессов предприятия для формирования единой информационной среды. Набор систем автоматизации и их настройка индивидуальны для каждого предприятия, что делает это направление более подходящим для частного облака. Чтобы осуществить базовую интеграцию, необходима более тщательная разработка иерархии пользовательских ролей для разграничения доступа, создания е единой базы данных пользователей и внедрения е технологии единого входа (single sign-on), э что обеспечит простую навигацию пользователей между системами. Более сложная интеграция заключатся в обмене данными между системами. Например, номенклатура моделей и обуви может быть автоматически выгружена из САПР в систему управления ресурсами предприятия, далее эти

данные передаются в систему электронного документооборота, которая проводит эмиссию в информационной системе маркировки и прослеживания «Честный знак».

Отдельным направлением является совершенствование процессов удаленной работы. В настоящее время тема удаленной работы приобретает все большую популярность. В 2020 году мировую экономику потрясло воздействие COVID-19 и введение карантинных мер, сдерживающих его. Ограничительные меры на передвижения и социальные контакты привело к тому, что большинство производств было приостановлено, а множество компаний во всем мире были вынуждены закрыться. Особенно эта проблема близка представителям малого и среднего бизнеса, которые в большей степени подвержены воздействию негативных факторов. Также ситуация осложнилась невозможностью точного определения длительности ограничений, которые введены на уровне отдельных территориальных единиц. Облачная САПР обуви обладает возможностью выполнения работы без привязки к своему рабочему месту, однако веб-технологии не учитывают свойства устройства, с которого выполняется работа. Адаптация клиентского приложения под мобильные устройства позволит увеличить количество вариантов рабочих мест.

Описанные направления развития реализуются путем создания небольших программ-сервисов. При решении новых задач могут применяться любые технологии, благодаря независимости сервисов друг от друга. В свою очередь, интеграция сторонних систем осуществляется через создание сервиса-адаптера, способного конвертировать передаваемые сигналы и данные в интегрируемую систему. Таким образом, разработанная концепция позволяет сосредоточиться на развитии облачной САПР обуви и сопутствующих исследованиях.

Выводы по третьей главе

1. Предложена реализация ядра системы и разработаны:
   - подсистемы ядра: движок, конфигурация, репозитории локальных и общих временных файлов, репозиторий сервисов, реестры внутренних и внешних сервисов, подсистема запуска модулей, обработчик запросов, контейнер очереди запросов, сервер, логер, подсистемы мониторинга и обеспечения контроля доступа;
   - пошаговый процесс осуществления аутентификации входящего запроса;
   - структуры данных для описания модулей и пересылаемых сообщений;
   - формы конфигурации ядра и сервисов, описываемых текстовым форматом JSON;
   - алгоритмы синхронной и асинхронной обработки входящих сообщений, обработки собственно сообщения, формирования и отправки ответного сообщения.
2. Разработан модуль оцифровки чертежа с использованием средств технического зрения, для которого

предложены:

   - применение камеры смартфона для получение исходного изображения чертежа;
   - этапы оцифровки – калибровка камеры, подготовка изображения, векторизация;
   - метод калибровки камеры при помощи открытой библиотеки обработки изображений OpenCV;
   - метод подготовки изображения к оцифровке с использованием реперных точек и библиотеки OpenCV;

разработаны:

- сервис управления камерами, выполняющий калибровку камеры, и осуществляющий контроль за данными, полученными в результате калибровки;
- сервис оцифровки, реализующий предложенный метод подготовки изображения к обработке;
- алгоритмы обработки сообщений модулями управления камерами и оцифровки.

3. Разработана модель проектных данных, иерархия которой основана на уровнях абстракции. Предложено, для компактного хранения информации в базе данных, в качестве основного графического примитива использовать сплайн Catmull-Rom, который может быть преобразован в множество кубических кривых Безье и обратно, что обеспечивает конвертируемость между моделью проектных данных, формата векторизованного изображения и стандартов отображения графических элементов на экране.
4. Рассмотрев современные базы данных, их достоинства и недостатки, предложено использовать объектно-реляционную систему управления базами данных, например, PostgreSQL, которая позволяет хранить как реляционные, так и нереляционные данные.
5. Представлены направления развития предлагаемой концепции, включающей: подготовительный этап – построение инфраструктуры; основной этап – развитие облачной САПР обуви по: совершенствованию проектирования; ритейл-направлению - более тесное взаимодействие с покупателями; корпоративному направлению - создание единой информационной среды предприятия.

ОБЩИЕ ВЫВОДЫ ПО РАБОТЕ

1. Рассмотрена проблема удаленной работы в системе автоматизированного проектирования обуви. Показано, что стоимость системы, ее привязка к аппаратному ключу и низкая эффективность существующих методов оцифровки являются препятствиями для применения САПР обуви в режиме удаленного доступа. Проанализирован текущий уровень состояния современных САПР с позиций возможностей облачных технологий. Установлено, что существующие облачные САПР имеют машиностроительную направленность и не могут быть применены для проектирования обуви.
2. Показано, что достоинствами облачных САПР общего назначения перед десктопными являются кроссплатформенность, быстрое развертывание и обновление, отсутствие необходимости содержать аппаратный ключ защиты, простая интеграция с другими системами, меньшая стоимость, возможность применения технологий, требовательных к вычислительной мощности, которые позволяют повысить эффективность процесса оцифровки и решить проблему удаленного применения САПР обуви.
3. Дано общее представление об облачной САПР обуви: основные участники, их связь с моделями обслуживания и развертывания системы. Для обеспечения эффективного внедрения облачных технологий в САПР обуви и ее работы в режиме удаленного доступа предложена концепция, включающая разработку архитектуры системы, допускающая оцифровку и чертежей с использованием технического зрения.
4. Рассмотрены: понятие архитектуры, критерии, влияющие на процессы жизненного цикла системы, а также критерии качества. Показана необходимость модульного подхода при проектировании архитектуры. Предложены возможные модули облачной САПР обуви: управления

пользователями и проектами, моделирования, градирования, раскроя, импорта, оцифровки, экспорта.

5. Проведено проектирование архитектуры облачной САПР обуви на основе шаблонов монолитной, микроядерной, сервис-ориентированной архитектур, что позволило предложить гибридную архитектуру облачной САПР обуви, которая в качестве компонентов содержит модели: ядра микроядерной архитектуры, модуля микросервисной архитектуры, взаимодействия сервис-ориентированной архитектуры. Показано, что изменение ядра системы и развертывание инфраструктурных компонентов позволяет с минимальными трудозатратами перестроить общую форму архитектуры.
6. Предложена реализация ядра системы и разработаны:
   - подсистемы ядра: движок, конфигурация, репозитории локальных и общих временных файлов, репозиторий сервисов, реестры внутренних и внешних сервисов, подсистема запуска модулей, обработчик запросов, контейнер очереди запросов, сервер, логер, подсистемы мониторинга и обеспечения контроля доступа;
   - пошаговый процесс осуществления аутентификации входящего запроса;
   - структуры данных для описания модулей и пересылаемых сообщений;
   - модели конфигурации ядра и сервисов, описываемых текстовым форматом JSON;
   - алгоритмы синхронной и асинхронной обработки входящих сообщений, обработки собственно сообщения, формирования и отправки ответного сообщения.
7. Разработан модуль оцифровки чертежа обуви с использованием средств технического зрения, для которого:

предложен метод оцифровки, включающий

- этапы оцифровки - калибровка камеры, подготовка изображения, векторизация;

- способ калибровки камеры;
- алгоритм подготовки изображения к оцифровке с использованием реперных точек и библиотеки OpenCV;

разработан:

- сервис управления камерами, выполняющий калибровку камеры, и осуществляющий контроль за данными, полученными в результате калибровки;
- сервис оцифровки, реализующий предложенный метод подготовки изображения к обработке;
- алгоритмы обработки сообщений модулями управления камерами и оцифровки.

8. Разработана модель проектных данных, иерархия которой основана на уровнях абстракции. Предложено использовать в качестве основного графического примитива использовать сплайн Catmull-Rom для компактного хранения информации в базе данных и обеспечения конвертируемости между моделью проектных данных и сторонними системами.
9. Рассмотрев современные базы данных, их достоинства и недостатки, предложено использовать объектно-реляционную систему управления базами данных, например, PostgreSQL, которая позволяет хранить как реляционные, так и нереляционные данные.
10. Отдельные положения диссертации прошли апробацию в условиях АО «Егорьевск-обувь», результаты которой подтверждают не только эффективность предложенного метода оцифровки, но и модуля в целом.
11. Представлены направления развития предлагаемой концепции, включающие: совершенствование проектирования при построении инфраструктуры и эволюции облачной САПР обуви; ритейл-ориентацию - более тесное взаимодействие с покупателями; вектор корпоративности - создание единой информационной среды предприятия

СПИСОК ЛИТЕРАТУРЫ

1. Анализ рынка обуви: рейтинг производителей обуви в России. — Текст : электронный // Деловой профиль : [сайт]. — URL: https:// delprof.ru/ press-center/ open-analytics/ analiz-rynka-obuvi-reyting-proizvoditeley-obuvi-v-rossii-/ (дата обращения: 20.03.2021).
2. Стратегия развития легкой промышленности в Российской Федерации на период до 2025 года. — Текст : электронный // Министерство промышленности и торговли России : [сайт]. — URL: https:// minpromtorg.gov.ru/ common/ upload/ files/ docs/ 260615-Ctrategiya_ITOG.docx (дата обращения: 15.11.2021).
3. Указ о национальных целях развития России до 2030 года. — Текст : электронный // Администрация Президента России : [сайт]. — URL: http:// kremlin.ru/ events/ president/ news/ 63728 (дата обращения: 15.11.2021).
4. Гусев А. О., Костылева В. В., Разин И. Б. Направления развития облачных САПР // Инновационное развитие легкой и текстильной промышленности (ИНТЕКС-2019). – 2019. – С. 183-186.
5. Гусев А. О., Костылева В. В., Разин И. Б. Облачные технологии и САПР в индустрии моды. // Сборник статей международной научно-практической конференции «Фундаментальные и прикладные проблемы создания материалов и аспекты технологий текстильной и легкой промышленности». – 2019, с. 277-279.
6. Wu D. et al. Towards a cloud-based design and manufacturing paradigm: looking backward, looking forward // International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. – American Society of Mechanical Engineers, 2012. – Т. 45011. – С. 315-328.

7. О Стратегии национальной безопасности Российской Федерации. Указ Президента Российской Федерации от 31 декабря 2015 года № 683. — Текст : электронный // Администрация Президента России : [сайт]. — URL: http:// kremlin.ru/ acts/ bank/ 40391 (дата обращения 03.09.2021)
8. Стратегия развития информационного общества в Российской Федерации на 2017 - 2030 годы. — Текст : электронный // Администрация Президента России : [сайт]. — URL: http:// www.kremlin.ru/ acts/ bank/ 41919/ page/ 2 (дата обращения 03.09.2021)
9. Указ Президента РФ от 1 декабря 2016 г. № 642 «О Стратегии научно-технологического развития Российской Федерации». — Текст : электронный // Администрация Президента России : [сайт]. — URL: http:// kremlin.ru/ acts/ bank/ 41449 (дата обращения 03.09.2021)
10. Vansevicius A. Cloud-based technologies in technical drawing // Journal Biuletyn of Polish Society for Geometry and Engineering Graphics. – 2019. – Т. 32. – С. 35--38.
11. Махалина О. М., Махалин В. Н. Своевременные и адекватные меры по поддержке экономики и населения России в период пандемии // Вестник РГГУ. Серия «Экономика. Управление. Право». – 2021. – №. 1. – С. 56-70.
12. Дёжкин М. Ю., Смирнова Т. В. Особенности дистанционной работы в пандемийный период // Вестник науки. – 2021. – Т. 4. – №. 10. – С. 78-88.
13. «Делойт» в СНГ и X5 Group выяснили, сколько компаний оставили сотрудников на удаленной работе. — Текст : электронный // Deloitte : [сайт]. — URL: https:// www2.deloitte.com/ ru/ ru/ pages/ about-deloitte/ press-releases/ 2021/ remote-work.html (дата обращения: 15.11.2021).
14. Райффайзенбанк: Россиянам на удаленке не хватает движения и разговоров с коллегами. — Текст : электронный // Райффайзенбанк : [сайт]. — URL:

https:// www.raiffeisen.ru/ about/ press/ releases/ 190490/ (дата обращения: 15.11.2021).

15. Санникова О. В. Некомпенсируемая образовательная миграция как проблема развития российского региона // Теория и практика общественного развития. – 2015. – №. 24.
16. Hirz M. et al. Integrated computer-aided design in automotive development // Springer-Verlag, Berlin-Heidelberg, DOI. – 2013. – Т. 10. – С. 978-3.
17. Чухарев Н. В., Комялова Е. В. САПР для машиностроения // Научно-техническое и экономическое сотрудничество стран АТР в XXI веке. – 2011. – Т. 3. – С. 229-232.
18. Zhuming Bi, Xiaoqin Wang Computer Aided Design and Manufacturing. – John Wiley & Sons, 2020.
19. САПР для обуви. — Текст : электронный // Яндекс.Дзен : [сайт]. — URL: https:// zen.yandex.ru/ media/ proshoes/ sapr-dlia-obuvi-5c4f01610d446800ad3ac2e7 (дата обращения: 25.05.2021).
20. Каган В. М. АСКО-2Д. — Текст : электронный // АСКО-2Д : [сайт]. — URL: http:// asko2d.narod.ru (дата обращения: 25.05.2021).
21. Ассоль обувь. — Текст : электронный // Центр Ассоль : [сайт]. — URL: https:// assol.org (дата обращения: 25.05.2021).
22. Autodesk ArtCAM Software End-of-Sale & End-of-Development FAQs. — Текст : электронный // Autodesk : [сайт]. — URL: https:// knowledge.autodesk.com/ search-result/ caas/ sfdcarticles/ sfdcarticles/ Autodesk-ArtCAM-Software-End-of-Sale-End-of-Development-FAQs.html (дата обращения: 25.05.2021).

23. Ильюшин С. В. Разработка методики проектирования обуви в формате 3-D с использованием технологий обратного инжиниринга // Сборник тезисов: Казанский государственный технологический университет. – 2014. – С. 76.
24. Сравнение подписки AutoCAD. — Текст : электронный // Autodesk : [сайт]. — URL: https:// www.autodesk.ru/ products/ autocad/ compare (дата обращения: 26.05.2021).
25. Выполнение поставки ПО САПР «АСКО-2Д». — Текст : электронный // Система Контур. Закупки : [сайт]. — URL: https:// zakupki. kontur.ru/ 31300337608 (дата обращения: 26.05.2021).
26. Гусев А. О., Костылева В. В., Разин И. Б. О направлениях развития обувных САПР // Сборник методических рекомендаций по вопросам развития технических и естественных наук. – 2019. – С. 10-15.
27. Гусев А. О., Костылева В. В., Разин И. Б. Анализ текущего этапа развития обувных САПР. // Сборник научных трудов научно-практической конференции «Эргодизайн как инновационная технология проектирования изделий и предметно-пространственной среды: инклюзивный аспект. Часть 1. – М.: РГУ им. А.Н. Косыгина, 2019, с. 83-87
28. Software for shoe designers. — Текст : электронный // Arsutoria School : [сайт]. — URL: https:// www.arsutoriaschool.com/ software-for-shoe-designers-and-technical-drawing/ (дата обращения: 26.05.2021).
29. Ishwor Khadka Software piracy: A study of causes, effects and preventive measures. Helsinki Metropolia University of Applied Sciences, 2015.
30. Кравченко А. С., Сахаров С. Л. Применение аппаратных ключей для защиты программного обеспечения // Вестник Воронежского института ФСИН России. – 2015. – №. 2. – С. 38-40.

31. 1С:Система лицензирования и защиты конфигураций. — Текст : электронный // 1С : [сайт]. — URL: https:// solutions.1c.ru/ catalog/ slk/ usb (дата обращения: 25.05.2021).
32. Клиент потерял ключ от программы 1С. — Текст : электронный // Центр АСП : [сайт]. — URL: http:// asp-partners.ru/ useful_info/ faq/ index.php? ELEMENT_ID=75171 (дата обращения: 25.05.2021).
33. Муртазина А. Р., Разин И. Б. Программное обеспечение для векторизации чертежей // Сборник научных трудов аспирантов. – 2014. – С. 70-75.
34. Муртазина А. Р. и др. Концепция модуля «оцифровка» в системах автоматизированного проектирования конструкций верха обуви // Дизайн и технологии. – 2016. – №. 54. – С. 30-35.
35. Lasi H. et al. Industry 4.0 // Business & information systems engineering. – 2014. – Т. 6. – №. 4. – С. 239-242.
36. Dalenogare L. S. et al. The expected contribution of Industry 4.0 technologies for industrial performance // International Journal of Production Economics. – 2018. – Т. 204. – С. 383-394.
37. Bertola P., Teunissen J. Fashion 4.0. Innovating fashion industry through digital transformation // Research Journal of Textile and Apparel. – 2018.
38. Mell P. et al. The NIST definition of cloud computing. — Текст : электронный // NIST : [сайт]. — URL: https:// csrc.nist.gov/ publications/ detail/ sp/ 800-145/ final (дата обращения: 25.05.2021).
39. Мурзин Ф. А., Батура Т. В., Семич Д. Ф. Облачные технологии: основные модели, приложения, концепции и тенденции развития // Программные продукты и системы. – 2014. – №. 3 (107).
40. Кантеева Ю. А., Порошина Л. А. Облачные технологии: история появления, виды, достоинства и недостатки. Критерии выбора «облака» //

Материалы секционных заседаний 60-й студенческой научно-практической конференции ТОГУ. – 2020. – С. 147-151.

41. Смирнов Ю. В. Облачные вычисления: история и влияние на будущее библиотек // Научные и технические библиотеки. – 2016. – №. 6. – С. 62-73.
42. Андреевский И.Л. Технологии облачных вычислений — СПб. : Санкт-Пб государственный экономический университет, 2018.
43. Радченко Е. П. Облачные технологии. Основные принципы, достоинства и недостатки // Интеллектуальные ресурсы-региональному развитию. – 2018. – №. 1. – С. 247.
44. Пурнак И. В., Самолова В. О. Применение облачных технологий в условиях современного бизнеса // Весенние дни науки. – 2020. – С. 722-729.
45. Sverrisdottir H. S., Ingason H. T., Jonasson H. I. The role of the product owner in scrum-comparison between theory and practices // Procedia-Social and Behavioral Sciences. – 2014. – Т. 119. – С. 257-267.
46. Bauer E., Adams R. Reliability and availability of cloud computing. – John Wiley & Sons, 2012.
47. Khalil I. M., Khreishah A., Azeem M. Cloud computing security: A survey // Computers. – 2014. – Т. 3. – №. 1. – С. 1-35.
48. CRN // Did Google's Energy-Efficient Battery Backups Put European Cloud Data At Risk? URL: https:// www.crn.com/ news/ cloud/ 300077854/ did-googles-energy-efficient-battery-backups-put-european-cloud-data-at-risk.htm (дата обращения: 21.08.2021).
49. ПО САПР для онлайн-проектирования. — Текст : электронный // AutoCAD : [сайт]. — URL: https:// www.autodesk.ru/ solutions/ cloud-based-online-cad-software (дата обращения: 26.05.2021).

50. Cloud Solutions by Siemens: Your Way, Your Pace. — Текст : электронный // Siemens: [сайт]. — URL: https:// www.sw.siemens. com/ en-US/ digital-transformation/ cloud/ (дата обращения: 26.05.2021).
51. One Cloud Platform, Endless Possibilities. — Текст : электронный // Dassault Systemes : [сайт]. — URL: https:// www.3ds.com/ 3dexperience/ cloud (дата обращения: 26.05.2021).
52. PTC gets more than «CLOUD CAD» with OnShape. — Текст : электронный // Tech-Clarity. : [сайт]. — URL: https:// tech-clarity.com/ cad-cloud/ 8318 (дата обращения: 26.05.2021).
53. PTC has Acquired Onshape. — Текст : электронный // PTC : [сайт]. — URL: https:// www.ptc.com/ en/ about/ onshape (дата обращения: 26.05.2021).
54. Is there a way to make a model from drawing? — Текст : электронный // OnShape : [сайт]. — URL: https:// forum.onshape.com/ discussion/ 659/ is-there-a-way-to-make-a-model-from-drawing (дата обращения: 21.08.2021).
55. Дизайнер-модельер обуви (в Китай). — Текст : электронный // HeadHunter : [сайт]. — URL: https:// hh.ru/ vacancy/ 12827715 (дата обращения: 26.05.2021).
56. Ladan M. Web services: technologies and benefits // Journal of Communication and Computer. – 2010. – Т. 7. – №. 6. – С. 31-37.
57. De Andrade P. R. M. et al. Cross platform app: a comparative study // arXiv preprint arXiv:1503.03511. – 2015.
58. Heitkötter H., Hanschke S., Majchrzak T. A. Evaluating cross-platform development approaches for mobile applications // International Conference on Web Information Systems and Technologies. – Springer, Berlin, Heidelberg, 2012. – С. 120-138.

59. Kell S. The operating system: should there be one? // Proceedings of the Seventh Workshop on Programming Languages and Operating Systems. – 2013. – С. 1-7.
60. Urma R. G., Fusco M., Mycroft A. Java 8 in action. – Manning publications, 2014.
61. Freeman A. Understanding the ASP. NET Core Platform // Pro ASP. NET Core 3. – Apress, Berkeley, CA, 2020. – С. 253-278.
62. Xanthopoulos S., Xinogalos S. A comparative analysis of cross-platform development approaches for mobile applications // Proceedings of the 6th Balkan Conference in Informatics. – 2013. – С. 213-220.
63. Lei K., Ma Y., Tan Z. Performance comparison and evaluation of web development technologies in php, python, and node. js // 2014 IEEE 17th international conference on computational science and engineering. – IEEE, 2014. – С. 661-668.
64. Tornincasa S. et al. The future and the evolution of CAD // Proceedings of the 14th international research/ expert conference: trends in the development of machinery and associated technology. – 2010. – Т. 1. – №. 1. – С. 11-18.
65. Кузина В. В., Глумскова И. О. Об особенностях применения операционной системы Astra Linux // Вестник ПГУАС: строительство, наука и образование. – 2020. – №. 2. – С. 93-96.
66. Пудовик Б. Р., Пушкаревский Ю. С. Особенности проектирования изделий с использованием систем автоматизированного проектирования // Системы управления и обработки информации. – 2018. – №. 4. – С. 86-90.
67. Wang P. et al. Generating precise dependencies for large software // 2013 4th International Workshop on Managing Technical Debt (MTD). – IEEE, 2013. – С. 47-50.

68. Celerier J. M. A cross-platform development toolchain for jit-compilation in multimedia software // Proceedings of the 17th Linux Audio Conference. – 2019.
69. Kula R. G. et al. Do developers update their library dependencies? // Empirical Software Engineering. – 2018. – Т. 23. – №. 1. – С. 384-417.
70. Bruun H. P. L. et al. PLM system support for modular product development // Computers in Industry. – 2015. – Т. 67. – С. 97-111.
71. Meier U. et al. Twenty Years of PLM–the Good, the Bad and the Ugly // IFIP International Conference on Product Lifecycle Management. – Springer, Cham, 2017. – С. 69-77.
72. Stark J. Product lifecycle management (volume 2): the devil is in the details. – Springer, 2015.
73. Суровцева О. А., Шишкина Г. И., Мереуц К. И. Разработка интегрированной системы для обувных предприятий // Молодежь и XXI век-2017. – 2017. – С. 374-377.
74. Rani D., Ranjan R. K. A comparative study of SaaS, PaaS and IaaS in cloud computing // International Journal of Advanced Research in Computer Science and Software Engineering. – 2014. – Т. 4. – №. 6.
75. Goyal S. Public vs private vs hybrid vs community-cloud computing: a critical review // International Journal of Computer Network and Information Security. – 2014. – Т. 6. – №. 3. – С. 20.
76. Raj C. P. R., Tolety S. B. A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach // 2012 Annual IEEE India Conference (INDICON). – IEEE, 2012. – С. 625-629.
77. Гусев А. О., Костылева В. В., Разин И. Б. Преимущества и недостатки построения клиентской части веб-приложения на основе модульного подхода // Сборник научных трудов Международной научно-практической

конференции «Фундаментальные и прикладные научные исследования в области инклюзивного дизайна и технологий: опыт, практика и перспективы. Часть 2. – М.: РГУ им. А.Н. Косыгина, 2021, с. 10-13

78. Гусев А. О., Костылева В. В., Разин И. Б. Сравнение технологий рендеринга для облачной САПР обуви // Дизайн, технологии и инновации в текстильной и легкой промышленности (ИННОВАЦИИ-2020): сборник материалов Международной научно-технической конференции. Часть 1.– М.: ФГБОУ ВО «РГУ им. АН Косыгина», 2020.–271 с. – С. 158.
79. Zissis D. et al. Collaborative CAD/ CAE as a cloud service // International Journal of Systems Science: Operations & Logistics. – 2017. – Т. 4. – №. 4. – С. 339-355.
80. Behr J. et al. A scalable architecture for the HTML5/ X3D integration model X3DOM // Proceedings of the 15th International Conference on Web 3D Technology. – 2010. – С. 185-194.
81. Parisi T. Programming 3D Applications with HTML5 and WebGL: 3D Animation and Visualization for Web Pages. – "O'Reilly Media, Inc.", 2014.
82. Leung C., Salga A. Enabling webgl // Proceedings of the 19th international conference on World wide web. – 2010. – С. 1369-1370. Miller C. A. et al. Scribl: an HTML5 Canvas-based graphics library for visualizing genomic data over the web // Bioinformatics. – 2013. – Т. 29. – №. 3. – С. 381-383.
83. Barrie J. et al. Applications for cloud-based CAD in design education and collaboration // DS 83: Proceedings of the 18th International Conference on Engineering and Product Design Education (E&PDE16), Design Education: Collaboration and Cross-Disciplinarity, Aalborg, Denmark, 8th-9th September 2016. – 2016. – С. 178-183.

84. Van Steen M., Pierre G., Voulgaris S. Challenges in very large distributed systems // Journal of Internet Services and Applications. – 2012. – Т. 3. – №. 1. – С. 59-66.
85. Bailis P., Kingsbury K. The network is reliable: An informal survey of real-world communications failures // Queue. – 2014. – Т. 12. – №. 7. – С. 20-32.
86. Косяков М.С. Введение в распределенные вычисления. – СПб: НИУ ИТМО, 2014. – 155 с.
87. Maddox P. Testing a distributed system // Communications of the ACM. – 2015. – Т. 58. – №. 9. – С. 54-58.
88. Wu J. Distributed system design. – CRC press, 2017.
89. Hashmi S. I. et al. Using the cloud to facilitate global software development challenges // 2011 IEEE Sixth International Conference on Global Software Engineering Workshop. – IEEE, 2011. – С. 70-77.
90. Kouroshfar E. et al. A Study on the Role of Software Architecture in the Evolution and Quality of Software // 2015 IEEE/ ACM 12th Working Conference on Mining Software Repositories. – IEEE, 2015. – С. 246-257.
91. Антамошкин О. Программная инженерия. Теория и практика. – Litres, 2019.
92. Bosch J. Speed, data, and ecosystems: the future of software engineering // IEEE Software. – 2015. – Т. 33. – №. 1. – С. 82-88.
93. Alenezi M. Software architecture quality measurement stability and understandability // International Journal of Advanced Computer Science and Applications (IJACSA). – 2016. – Т. 7. – №. 7. – С. 550-559.
94. Taylor J. T., Taylor W. T. Software architecture // Patterns in the Machine. – Apress, Berkeley, CA, 2021. – С. 63-82.

95. Richards M. Software architecture patterns. – 1005 Gravenstein Highway North, Sebastopol, CA 95472 : O'Reilly Media, Incorporated, 2015. – T. 4.
96. Basha N. M. J., Moiz S. A. Component based software development: A state of art // IEEE-international conference on advances in engineering, science and management (ICAESM-2012). – IEEE, 2012. – C. 599-604.
97. Beck F., Diehl S. On the congruence of modularity and code coupling // Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. – 2011. – C. 354-364.
98. Li Z. et al. An empirical investigation of modularity metrics for indicating architectural technical debt // Proceedings of the 10th international ACM Sigsoft conference on Quality of software architectures. – 2014. – C. 119-128.
99. Bucchiarone A. et al. From monolithic to microservices: An experience report from the banking domain // Ieee Software. – 2018. – T. 35. – №. 3. – C. 50-55.
100. Villamizar M. et al. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud // 2015 10th Computing Colombian Conference (10CCC). – IEEE, 2015. – C. 583-590.
101. Kambalyal C. 3-tier architecture // Retrieved On. – 2010. – T. 2. – №. 34. – C. 2010.
102. Barabanova I. A. et al. Analysis of the intermediate layer work in the three-tier architecture “client-server” of automation engineering problems // IOP Conference Series: Materials Science and Engineering. – IOP Publishing, 2019. – T. 537. – №. 3. – C. 032011.
103. Martin R. C., Grenning J., Brown S. Clean architecture: a craftsman's guide to software structure and design. – Prentice Hall, 2018. – C. 352.
104. Boukhary S., Colmenares E. A Clean Approach to Flutter Development through the Flutter Clean Architecture Package // 2019 International Conference on

Computational Science and Computational Intelligence (CSCI). – IEEE, 2019. – С. 1115-1120.

105. Gouigoux J. P., Tamzalit D. From monolith to microservices: Lessons learned on an industrial migration to a web oriented architecture // 2017 IEEE International Conference on Software Architecture Workshops (ICSAW). – IEEE, 2017. – С. 62-65.
106. Chen R., Li S., Li Z. From monolith to microservices: A dataflow-driven approach // 2017 24th Asia-Pacific Software Engineering Conference (APSEC). – IEEE, 2017. – С. 466-475.
107. Arango E. C., Loaiza O. L. SCRUM Framework Extended with Clean Architecture Practices for Software Maintainability // Computer Science On-line Conference. – Springer, Cham, 2021. – С. 667-681.
108. Troubitsyna E., Javed K. Towards systematic design of adaptive fault tolerant systems // ADAPTIVE 2014, The Sixth International Conference on Adaptive and Self-Adaptive Systems and Applications. – 2014.
109. Martens A. et al. From monolithic to component-based performance evaluation of software architectures // Empirical Software Engineering. – 2011. – Т. 16. – №. 5. – С. 587-622.
110. Pessoa T. et al. An eclipse plugin to support code smells detection // arXiv preprint arXiv:1204.6492. – 2012.
111. Michel L., Van Hentenryck P. A microkernel architecture for constraint programming // Constraints. – 2017. – Т. 22. – №. 2. – С. 107-151.
112. Setapa S. et al. Trusted computing based microkernel // 2010 International Conference on Computer Applications and Industrial Electronics. – IEEE, 2010. – С. 1-4.

113. Lee T. Y., Seo H. R., Shin D. R. Analysis of component model by kernel structures // 2010 International Conference on Electronics and Information Engineering. – IEEE, 2010. – T. 2. – C. V2-45-V2-48.

114. Wang S., Ji Y., Yang S. A Micro-Kernel Test Engine for Automatic Test System // J. Comput. – 2011. – T. 6. – №. 1. – C. 3-10.

115. Pohle A. et al. Capability wrangling made easy: debugging on a microkernel with Valgrind // ACM Sigplan Notices. – 2010. – T. 45. – №. 7. – C. 3-12.

116. Hlavatovič A., Krajčovič T. Microkernel system as basis for system library based on generic components // International Conference February. – 2010. – T. 10. – C. 13.

117. Lampka K., Lackorzynski A. Using hypervisor technology for safe and secure deployment of high-performance multicore platforms in future vehicles // 2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS). – IEEE, 2019. – C. 783-786.

118. Weinhold C. et al. FFMK: A fast and fault-tolerant microkernel-based system for exascale computing // Software for Exascale Computing-SPPEXA 2013-2015. – Springer, Cham, 2016. – C. 405-426.

119. Mi Z. et al. Skybridge: Fast and secure inter-process communication for microkernels // Proceedings of the Fourteenth EuroSys Conference 2019. – 2019. – C. 1-15.

120. Mueller B. et al. Understanding the economic potential of service-oriented architecture // Journal of Management Information Systems. – 2010. – T. 26. – №. 4. – C. 145-180.

121. Rosen M. et al. Applied SOA: service-oriented architecture and design strategies. – John Wiley & Sons, 2012.

122. Zhu J. et al. Scaling service-oriented applications into geo-distributed clouds // 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering. – IEEE, 2013. – С. 335-340.
123. Jiang J. et al. Modeling and analyzing mixed communications in service-oriented trustworthy software // Science China Information Sciences. – 2012. – Т. 55. – №. 12. – С. 2738-2756.
124. Czaja L. Remote procedure call // Introduction to distributed computer systems. – Springer, Cham, 2018. – С. 141-155.
125. Adamczyk P. et al. Rest and web services: In theory and in practice // REST: from research to practice. – Springer, New York, NY, 2011. – С. 35-57.
126. Schreier S. Modeling restful applications // Proceedings of the second international workshop on restful design. – 2011. – С. 15-21.
127. Vogel M., Weber S., Zirpins C. Experiences on migrating RESTful web services to GraphQL // International Conference on Service-Oriented Computing. – Springer, Cham, 2017. – С. 283-295.
128. Gessert F. et al. Towards a Scalable and Unified REST API for Cloud Data Stores // GI-Jahrestagung. – 2014. – С. 723-734.
129. Гусев А. О., Костылева В. В., Разин И. Б. Сравнение алгоритмов балансировки нагрузки // Инновационное развитие техники и технологий в промышленности (ИНТЕКС-2020). – 2020. – С. 143-146.
130. Li L. et al. Design patterns and extensibility of REST API for networking applications // IEEE Transactions on Network and Service Management. – 2016. – Т. 13. – №. 1. – С. 154-167.
131. Indrasiri K., Kuruppu D. GRPC: Up and Running: Building Cloud Native Applications with Go and Java for Docker and Kubernetes. – " O'Reilly Media, Inc.", 2020.

132. Feng J., Li J. Google protocol buffers research and application in online game // IEEE conference anthology. – IEEE, 2013. – C. 1-4.
133. Loreto S. et al. Known issues and best practices for the use of long polling and streaming in bidirectional HTTP // Internet Engineering Task Force, Request for Comments. – 2011. – T. 6202. – №. 2070-1721. – C. 32.
134. Molchanov H., Zhmaiev A. Circuit breaker in systems based on microservices architecture // Advanced Information Systems. – 2018. – T. 2. – №. 4. – C. 74-77.
135. Heidari A., Navimipour N. J. Service discovery mechanisms in cloud computing: a comprehensive and systematic literature review // Kybernetes. – 2021.
136. Al Nuaimi K. et al. A survey of load balancing in cloud computing: Challenges and algorithms // 2012 second symposium on network cloud computing and applications. – IEEE, 2012. – C. 137-142.
137. Bak S. et al. Multi-level load balancing with an integrated runtime approach // 2018 18th IEEE/ ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID). – IEEE, 2018. – C. 31-40.
138. Tang K. et al. Design and implementation of push notification system based on the MQTT protocol // International Conference on Information Science and Computer Applications (ISCA 2013). – Atlantis Press, 2013. – C. 116-119.
139. Borangiu T. IBM service oriented technologies and management for smarter enterprise // 15th International Conference on System Theory, Control and Computing. – IEEE, 2011. – C. 1-6.
140. Tsai W. T., Sun X., Balasooriya J. Service-oriented cloud computing architecture // 2010 seventh international conference on information technology: new generations. – IEEE, 2010. – C. 684-689.

141. Sun C., El Khoury E., Aiello M. Transaction management in service-oriented systems: Requirements and a proposal // IEEE Transactions on services computing. – 2010. – Т. 4. – №. 2. – С. 167-180.
142. Dürr K., Lichtenthäler R., Wirtz G. An Evaluation of Saga Pattern Implementation Technologies // ZEUS. – 2021. – С. 74-82.
143. Rudrabhatla C. K. Comparison of event choreography and orchestration techniques in microservice architecture // International Journal of Advanced Computer Science and Applications. – 2018. – Т. 9. – №. 8. – С. 18-22.
144. Wei Y., Blake M. B. Service-oriented computing and cloud computing: Challenges and opportunities // IEEE Internet Computing. – 2010. – Т. 14. – №. 6. – С. 72-75.
145. Shamsoddin-Motlagh E. A survey of service oriented architecture systems testing // arXiv preprint arXiv:1212.3248. – 2012.
146. Bellomo S. Suggestions for documenting SOA-based systems. – Carnegie-Mellon, 2010.
147. Lloyd W. et al. Dynamic scaling for service oriented applications: implications of virtual machine placement on IaaS clouds // 2014 IEEE International Conference on Cloud Engineering. – IEEE, 2014. – С. 271-276.
148. Aazam M., Huh E. N. Cloud broker service-oriented resource management model // Transactions on Emerging Telecommunications Technologies. – 2017. – Т. 28. – №. 2. – С. e2937.
149. Wang S. et al. Towards an accurate evaluation of quality of cloud service in service-oriented cloud computing // Journal of Intelligent Manufacturing. – 2014. – Т. 25. – №. 2. – С. 283-291.

150. Salah T. et al. The evolution of distributed systems towards microservices architecture // 2016 11th International Conference for Internet Technology and Secured Transactions (ICITST). – IEEE, 2016. – С. 318-325.
151. Larrucea X. et al. Microservices // IEEE Software. – 2018. – Т. 35. – №. 3. – С. 96-100.
152. BERNET C. The cost of leadership: An investment in success // Industrial Management. – 2021. – Т. 63. – №. 1.
153. Hassan S., Bahsoon R. Microservices and their design trade-offs: A self-adaptive roadmap // 2016 IEEE International Conference on Services Computing (SCC). – IEEE, 2016. – С. 813-818.
154. Tonse S. Scalable microservices at Netflix. challenges and tools of the trade. – 2018.
155. Singleton A. The economics of microservices // IEEE Cloud Computing. – 2016. – Т. 3. – №. 5. – С. 16-20.
156. Pahl C., Jamshidi P., Zimmermann O. Microservices and containers // Software Engineering 2020. – 2020.
157. Pahl C. Containerization and the paas cloud // IEEE Cloud Computing. – 2015. – Т. 2. – №. 3. – С. 24-31.
158. Odun-Ayo I. et al. Virtualization, containerization, composition, and orchestration of cloud computing services // International Conference on Computational Science and Its Applications. – Springer, Cham, 2019. – С. 403-417.
159. Munaf R. M. et al. Microservices architecture: Challenges and proposed conceptual design // 2019 International Conference on Communication Technologies (ComTech). – IEEE, 2019. – С. 82-87.

160. Götz B. et al. Challenges of production microservices // Procedia CIRP. – 2018. – T. 67. – C. 167-172.
161. Song M., Zhang C., Haihong E. An auto scaling system for API gateway based on Kubernetes // 2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS). – IEEE, 2018. – C. 109-112.
162. Sayfan G. Mastering kubernetes. – Packt Publishing Ltd, 2017.
163. Indrasiri K., Siriwardena P. Service Mesh // Microservices for the Enterprise. – Apress, Berkeley, CA, 2018. – C. 263-292.
164. Burns B., Oppenheimer D. Design patterns for container-based distributed systems // 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16). – 2016.
165. Bhattacharya R. Smart proxying for microservices // Proceedings of the 20th International Middleware Conference Doctoral Symposium. – 2019. – C. 31-33.
166. Siriwardena P. Edge Security with an API Gateway // Advanced API Security. – Apress, Berkeley, CA, 2020. – C. 103-127.
167. Krawczyk H., Paterson K. G., Wee H. On the security of the TLS protocol: A systematic analysis // Annual Cryptology Conference. – Springer, Berlin, Heidelberg, 2013. – C. 429-448.
168. Duffield M. Binding Behaviors // Practical App Development with Aurelia. – Apress, Berkeley, CA, 2018. – C. 171-177.
169. Song M., Zhang C., Haihong E. An auto scaling system for API gateway based on Kubernetes // 2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS). – IEEE, 2018. – C. 109-112.
170. Messina A. et al. The database-is-the-service pattern for microservice architectures // International Conference on Information Technology in Bio-and Medical Informatics. – Springer, Cham, 2016. – C. 223-233.

171. Heorhiadi V. et al. Gremlin: Systematic resilience testing of microservices // 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS). – IEEE, 2016. – С. 57-66.

172. Форд Н., Парсонс Р., Куа П. Эволюционная архитектура. Поддержка непрерывных изменений. - СПб.: Питер, 2019. — 268 с.

173. Гусев А.О., Костылева В.В., Разин И.Б. Гибридная архитектура облачной САПР обуви // Сборник научных трудов Международного научно-технического симпозиума «Современные инженерные проблемы в производстве товаров народного потребления» III Международного Косыгинского Форума «Современные задачи инженерных наук». - М.: «РГУ им. А.Н. Косыгина», с. 225-228

174. Ihrig C. J. Executing Code // Pro Node. js for Developers. – Apress, Berkeley, CA, 2013. – С. 129-145.

175. Guan M., Gu M. Design and implementation of an embedded web server based on ARM // 2010 IEEE International Conference on Software Engineering and Service Sciences. – IEEE, 2010. – С. 612-615.

176. Choi J., Kim J., Han H. Efficient memory mapped file I/ O for in-memory file systems // 9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 17). – 2017.

177. Papagiannis A., Marazakis M., Bilas A. Memory-mapped I/ O on steroids // Proceedings of the Sixteenth European Conference on Computer Systems. – 2021. – С. 277-293.

178. Weeks N. T., Kraeva M., Luecke G. R. Parallelization of software pipelines using the mpififo Tool // 2013 IEEE International Conference on Cluster Computing (CLUSTER). – IEEE, 2013. – С. 1-3.

179. Palakollu S. M. Interprocess Communication // Practical System Programming with C. – Apress, Berkeley, CA, 2021. – С. 165-214.
180. Гукасян А. А. Идентификация аутентификация // Аллея науки. – 2018. – Т. 1. – №. 1. – С. 457-460.
181. Narendrakumar S. et al. Token security for internet of things // International Journal of Embedded Systems. – 2018. – Т. 10. – №. 4. – С. 334-343.
182. He X., Yang X. Authentication and authorization of end user in microservice architecture // Journal of Physics: Conference Series. – IOP Publishing, 2017. – Т. 910. – №. 1. – С. 012060.
183. Mundada Y., Feamster N., Krishnamurthy B. Half-baked cookies: Hardening cookie-based authentication for the modern web // Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security. – 2016. – С. 675-685.
184. Pezoa F. et al. Foundations of JSON schema // Proceedings of the 25th International Conference on World Wide Web. – 2016. – С. 263-27
185. А. Р. Муртазина, А. О. Гусев, В. В. Костылева, И. Б. Разин Векторизация чертежей конструкций верха обуви с использованием открытых библиотек // Дизайн и технологии. – 2019. – № 74(116). – С. 135-141.
186. Strategy Analytics: Half the World Owns a Smartphone // Business wire URL: https:// www.businesswire.com/ news/ home/ 20210624005926/ en/ Strategy-Analytics-Half-the-World-Owns-a-Smartphone (дата обращения: 28.07.2021).
187. Olympus Agreed on Transfer of Imaging Business with JIP // Olympus Europe URL: https:// www.olympus-europa.com/ company/ en/ news/ press-releases/ 2020-09-30t10-34-22/ olympus-agreed-on-transfer-of-imaging-business-with-jip-28352.html (дата обращения: 20.07.2021).

188. Гусев А.О., Костылева В. В., Разин И. Б., Белгородский В.С. Описание NURBS сплайнами контура изображения, векторизованного при помощи поливекторных полей // В сборнике Международного научно-технического симпозиума: Современные инженерные проблемы в производстве товаров народного потребления» Международного Косыгинского Форума. Часть 1. – М.: РГУ им. А.Н. Косыгина, 2019, с. 208-214
189. Шапиро Л. Компьютерное зрение: учебное пособие // Шапиро Л., Стокман Дж., пер. с англ. 4-е изд. — Москва: Лаборатория знаний, 2020.
190. Муртазина А. Р. Разработка системы проектирования конструкций верха обуви с использованием средств технического зрения: дис. ... канд. тех. наук: 05.19.05. - М., 2016.
191. Разин И. Б. и др. Применение библиотеки OpenCV в системах автоматизированного проектирования // Дизайн, технологии и инновации в текстильной и легкой промышленности (ИННОВАЦИИ-2015). – 2015. – С. 244-246.
192. Гусев С. С. Векторизация растровых изображений // Прикладная математика и вопросы управления. – 2018. – №. 4. – С. 83-98.
193. Гусев А. О., Костылева В.В., Разин И.Б., Муртазина А.Р. Сравнение производительности многопоточной реализации алгоритма скелетизации Зонга-Суня // Сборник научных трудов Международной научно-практической заочной конференции: Концепции, теория, методики фундаментальных и прикладных научных исследований в области инклюзивного дизайна и технологий. – 2020, с. 6-10.
194. 193 Eisenberg J. D., Bellamy-Royds A. SVG essentials: Producing scalable vector graphics with XML. – " O'Reilly Media, Inc.", 2014.

195. Alexander G., Botvin M. Comparative analysis and research of digital image encoding algorithms // InterConf. – 2020.

196. Lee Y. J., Lee Y. S., Kim Y. B. Design of Platform Independent 3D Shoe CAD System // Journal of Korea Multimedia Society. – 2006. – Т. 9. – №. 8. – С. 1010-1019.

197. Arasteh S. T., Kalisz A. Conversion Between Bezier and Catmull-Rom Splines // arXiv preprint arXiv:2011.08232. – 2020.

198. Farin G. Curves and surfaces for computer-aided geometric design: a practical guide. – Elsevier, 2014.

199. Hattab A., Ainsworth M. Performance of bézier curves rendering in web browsers. – 2013.

200. Гусев А.О., Костылева В.В., Разин И.Б., Муртазина А.Р. Контроль версий в облачной системе автоматизированного проектирования обуви // Костюмология, 2021 №1, https:// kostumologiya.ru/ PDF/ 04TLKL121.pdf

201. Meijer E., Bierman G. A co-relational model of data for large shared data banks // Communications of the ACM. – 2011. – Т. 54. – №. 4. – С. 49-58.

202. Мартин Фаулер, Прамодкумар Дж. Садаладж. NoSQL. Новая методология разработки нереляционных баз данных. - М.: И.Д.Вильямс, 2013. 192 с.

203. Kalid S. et al. Big-data NoSQL databases: A comparison and analysis of "Big-Table", "DynamoDB", and "Cassandra" // 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA). – IEEE, 2017. – С. 89-93.

204. Burdakov A. V., Grigorev U. A., Ploutenko A. D. Comparison of table join execution time for parallel DBMS and MapReduce // Artificial Intelligence and Applications Proceedings. – 2014.

205. Kemme B., Alonso G. Database replication: a tale of research across communities // Proceedings of the VLDB Endowment. – 2010. – Т. 3. – №. 1-2. – С. 5-12.

206. Chandra M. S. S., Rao K. R., Hussain M. A. An efficient scheme for facilitating secure data sharing in decentralized disruption tolerant networks // Indian Journal of Science and Technology. – 2016. – Т. 9. – №. 5. – С. 1-13.

207. Li C. et al. Making geo-replicated systems fast as possible, consistent when necessary // 10th USENIX Symposium on Operating Systems Design and Implementation ({OSDI} 12). – 2012. – С. 265-278.

208. Stonebraker M. et al. The end of an architectural era: It's time for a complete rewrite // Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker. – 2018. – С. 463-489.

209. Abouzeid A. et al. HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads // Proceedings of the VLDB Endowment. – 2009. – Т. 2. – №. 1. – С. 922-933.

210. DeCandia G. et al. Dynamo: Amazon's highly available key-value store // ACM SIGOPS operating systems review. – 2007. – Т. 41. – №. 6. – С. 205-220.

211. Karloff H., Suri S., Vassilvitskii S. A model of computation for mapreduce // Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms. – Society for Industrial and Applied Mathematics, 2010. – С. 938-948.

212. Редмон Э., Уилсон Д. Р. Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL. – М.: ДМК Пресс, 2013. – 384 с.

213. Tamboli A. Creating a REST Interface // Build Your Own IoT Platform. – Apress, Berkeley, CA, 2019. – С. 137-163.

214. Григорьев Ю.А. Анализ свойств баз данных NoSQL // Информатика и системы управления. – 2013. № 2. С. 3-13.

215. Topcu A. E., Rmis A. M. Analysis and evaluation of the riak cluster environment in distributed databases // Computer Standards & Interfaces. – 2020. – Т. 72. – С. 103452.

216. Sliusar V. V. et al. Usage of triggers for business process controlling in ERP systems // 2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus). – IEEE, 2018. – С. 1567-1570.

217. Strauch C. NoSQL databases. Lecture selected topics on software-technology ultra-large scale sites // Manuscript. Stuttgart Media University. – 2011.

218. Mohamed M. A., Altrafi O. G., Ismail M. O. Relational vs. nosql databases: A survey // International Journal of Computer and Information Technology. – 2014. – Т. 3. – №. 03. – С. 598-601.

219. Chodorow K. MongoDB: the definitive guide: powerful and scalable data storage. – " O'Reilly Media, Inc.", 2013.

220. Harrison G., Harrison M. MongoDB Architecture and Concepts // MongoDB Performance Tuning. – Apress, Berkeley, CA, 2021. – С. 13-32.

221. Асриянц С. В., Пономарёва Е. Д., Аветян Г. А. Хранение документов в базе данных на примере системы управления базами данных POSTGRESQL // Российская наука в современном мире. – 2017. – С. 68-70.

222. Truskowski W., Klewek R., Skublewska-Paszkowska M. Comparison of MySQL, MSSQL, PostgreSQL, Oracle databases performance, including virtualization // Journal of Computer Sciences Institute. – 2020. – Т. 16. – С. 279-284.

223. Лашманов А. Импортозамещение: риски и иллюзии // Открытые системы. СУБД. – 2015. – №. 1. – С. 34-35.

224. Vazquez Ortíz Y., Mier Pierre L., Sotolongo León A. R. Características no relacionales de PostgreSQL: incremento del rendimiento en el uso de datos JSON // Revista Cubana de Ciencias Informáticas. – 2016. – Т. 10. – С. 70-81.

225. Суманеев А. П. Анализ производительности реляционных и NoSQL СУБД: дис. ... магистра - СПб., 2017.

226. А. Р. Муртазина, А. О. Гусев, В. В. Костылева, И. Б. Разин. Концепция использования сетевой корпоративной системы и открытых библиотек для перехода к «умному предприятию». // Известия высших учебных заведений. Технология легкой промышленности. – 2021. – Т. 51. – №. 1. – С. 102-106.

227. Костылева В. В., Литвин Е.В., Разин И.Б., Смирнов Е.Е. Информационно-телекоммуникационные технологии в автоматизированном дистанционном подборе изделий. Фундаментальные и прикладные научные исследования в области инклюзивного дизайна и технологий: опыт, практика и перспективы / Сборник научных трудов Международной научно-практической конференции. Часть 1. – М.: РГУ им. А.Н. Косыгина, 2021. С. 161-164.

228. Свидетельство Федеральной службы по интеллектуальной собственности о государственной регистрации программы для ЭВМ «Вычислительный сервис облачной системы автоматизированного проектирования обуви Calceus» № 2021615416

ПРИЛОЖЕНИЕ А

Акт о внедрении результатов НИОКР «Цифровизация и обработка чертежей конструкций верха обуви с использованием средств технического зрения»

«УТВЕРЖДАЮ»

Проректор по научной работе
ФГБОУ ВО «РГУ им. А.Н. Косыгина»
_____ Кащеев О.В
«____» _____ 2020г.

«УТВЕРЖДАЮ»

Генеральный директор
АО «Егорьевск-обувь»
_____Сорокин С.В
«____» _____ 2020 г.

АКТ

о внедрении результатов выполнения НИОКР
«Цифровизация и обработка чертежей конструкций верха обуви с использованием средств технического зрения»

Настоящий акт составлен представителями АО «Егорьевск-обувь» и ФГБОУ ВО «РГУ им. А.Н. Косыгина» о том, что в ходе выполнения НИОКР на тему: «Цифровизация и обработка чертежей конструкций верха обуви с использованием средств технического зрения» (руководитель - к.т.н., доц. Муртазина А.Р.) получена следующая научно-техническая продукция:

- концепция построения сетевой корпоративной системы автоматизированного проектирования контуров шаблонов деталей и чертежей конструкций верха обуви с использованием средств технического зрения, включающая:
- методику ввода чертежей с бумажных носителей, используя доступные технические средства;
- алгоритмы обработки растровых изображений чертежей, обеспечивающих переход к физическим единицам измерения с требуемой точностью;
- методику преобразования растрового представления контуров шаблонов деталей и чертежей конструкций верха в векторное;
- математическую модель и алгоритмы представления замкнутых контуров шаблонов деталей обуви;
- алгоритм оперативного регулирования числа точек в контуре.

Практическую ценность работы составляют разработанный алгоритм и математическая модель представления контуров шаблонов и чертежей деталей обуви, программно-реализованный модифицированный алгоритм векторизации для интеллектуального распознавания чертежей деталей обуви. Разработанный механизм интеллектуального распознавания чертежей деталей обуви положительно отражается на процессах конструкторско-технологической подготовки производства, что подтверждено в ходе экспериментальной апробации на АО «Егорьевск-обувь» и предполагается использовать при проектировании новых конструкций и ассортимента детской обуви.

Представители АО «Егорьевск-обувь»

Евплов А.В. -директор по международной промышленной кооперации
Корсакова А.В. директор по производству
Ермакова А. Р. -нач. отдела разработки

Представители ФГБОУ ВО «РГУ им. А.Н. Косыгина»
доц., к.т.н. Муртазина А.Р.
проф., д.т.н. Костылева В. В.
доц., к.т.н. Разин И.В.
доц., к.т.н. Синева О.В.
асп. Гусев А.О.

ПРИЛОЖЕНИЕ Б

Акт об использовании результатов диссертационной работы

«УТВЕРЖДАЮ»
Проректор по науке
РГУ им. А.Н. Косыгина
_____ А.В. Силаков
«__» _____ 2021 г.

АКТ

об использовании результатов диссертационной работы Гусева А.О.

Настоящий акт составлен о том, что в рамках научного проекта (грант ректора РГУ им. А.Н. Косыгина 2019-2020 г. г.) «Цифровизация и обработка чертежей конструкций верха обуви с использованием средств технического зрения» аспирантом Гусевым А.О. предложены способ взаимодействия между модулями в конфигурации отечественных САПР и метод оцифровки чертежа по изображению с использованием мобильных цифровых устройств для обеспечения эффективной работы в условиях удаленного доступа. Указанное определяет отдельные положения его диссертации на тему: «Разработка концепции системы автоматизированного проектирования обуви с применением облачных технологий».

Предложенные решения предполагается использовать на АО «Егорьевск-обувь» при создании новых моделей детской обуви.

Руководитель проекта:
к.т.н., доц. Муртазина А.Р.

Исполнители:
д.т.н., проф. Костылева В. В.
к.т.н., доц. Разин И. Б.
асп. Гусев А. О.
инж. Смолянинов К. М.

От отдела НИР:

Гуторова Н.В.
Михеева Н.А.
Николаева Н.А.

ПРИЛОЖЕНИЕ В

Свидетельство о государственной регистрации программы для ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2021615416

Вычислительный сервис облачной системы автоматизированного проектирования обуви Calceus

Правообладатель: ***Гусев Александр Олегович (RU)***

Авторы: ***Гусев Александр Олегович (RU)***

Заявка № **2021614403**
Дата поступления **26 марта 2021 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***7 апреля 2021 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

Г.П. Ивлиев