Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления
Кафедра интеллектуальных информационных технологий

К защите допустить:

Заведующий кафедрой ИИТ

_____ Д. В. Шункевич

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
по дисциплине «Проектирование баз знаний»
на тему:

Среда разработки компонентов пользовательских интерфейсов ostis-систем

БГУИР КП 6-40 03 01 02 080 ПЗ

Студент гр. 321703 А. В. Титов
Руководитель Н. В. Зотов

Минск 2025

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

БЗ — база знаний.

JavaScript — высокоуровневый, динамически типизированный язык программирования, который в первую очередь используется для создания интерактивных и динамических веб-страниц.

IDE — integrated development environment (интегрированная среда разработки).

OSTIS — Open Semantic Technologies for Intelligent System Design (открытая семантическая технология для интеллектуальных систем).

SC — Semantic Code (семантический код).

SCg — Semantic Code Graphical (графический семантический код).

SCs — Semantic Code string (строковый семантический код).

TypeScript — строго типизированное надмножество языка JavaScript, разработанное компанией Microsoft.

HTML — стандартизированный язык гипертекстовой разметки документов для просмотра веб-страниц в браузере.

CSS — формальный язык декорирования и описания внешнего вида документа, написанного с использованием языка разметки.

NLP — Natural Language Processing (обработка естественного языка) — это область искусственного интеллекта, которая сосредотачивается на взаимодействии между компьютерами и людьми с использованием естественного языка. Основная цель NLP состоит в том, чтобы дать машинам возможность понимать, интерпретировать и генерировать человеческий язык в полезной и значимой форме.

UI — user interface, пользовательский интерфейс, совокупность визуальных и интерактивных элементов, через которые человек взаимодействует с программным обеспечением, устройством или системой.

C++ — компилируемый, статически типизированный язык программирования общего назначения.

ВВЕДЕНИЕ

Пользовательский интерфейс (ПИ, англ. - UI) — это уровень системы «человек–компьютер», на котором происходит взаимодействие составляющих этой системы. ПИ включает в себя программное и аппаратное обеспечение, данные, полученные от пользователя посредством аппаратных устройств ввода (таких, как клавиатуры, мыши, web-камеры, сенсорные экраны, микрофоны и т.д.), а также иные виды взаимодействий пользователя с крупными автоматизированными системами, такими, как морское судно или электростанция.

Среда разработки UI — это программная платформа или инструмент, объединяющий средства визуального проектирования, редактирования, тестирования и реализации графического интерфейса приложения, предназначенные для упрощения и ускорения процесса создания UI.

Целью данного курсового проекта является создание среды разработки компонентов UI ostis-систем.

Для выполнения поставленной цели необходимо решить следующие задачи:

1 Изучение существующих UI, сред разработки интерфейсов и выделение их основных преимуществ и недостатков.

2 Анализ и исправление существующих в базе знаний компонентов интерфейса.

3 Проектирование недостающих компонентов интерфейса и их загрузка в базу знаний.

4 Создание итерпретатора компонентов интерфейса, взятых из базы знаний.

5 Создание среды сборки интерфейсов.

Ожидаемым результатом курсового проекта является:

– набор компонентов интерфейса, загруженный в базу знаний;

– итерпретатор вышеописанных компонентов.

1 АНАЛИЗ СУЩЕСТВУЮЩИХ ПОЛЬЗОВАТЕЛЬСКИХ ИНТЕРФЕЙСОВ И СРЕД РАЗРАБОТКИ ПОЛЬЗОВАТЕЛЬСКИХ ИНТЕРФЕЙСОВ

1.1 Общий анализ

Ключевыми требованиями к пользовательскому интерфейсу являются:

– интуитивность и понятность;

– удобство взаймодействия;

– персонализация;

– наличие справки об предназначении n-ого компонента интерфейса.

Для выполнение этих требований UI должен быть адаптивным. На данный момент адаптивность достигается различными методами:

– предоставление пользователю множества опции, изменяющих взаимодействие с интерфейсов (настройка действий для жестов, уменьшение/увеличение шрифта, установка качества видео по умолчанию и т.п.);

– предоставление различных версий/шаблонов интерфейса (мобильная/десктопная версия сайта);

– ручная настройка определенных элементов интерфейса (изменение размера и расположение джойстика в играх, настройка чувствительности мыши).

Степень адаптивности интерфейса определяется средой, в которой был разработан этот интерфейс (может ли интерфейс при необходимости обратиться к среде и запросить изменение своей структуры). Современные среды разработки пользовательских интерфейсов (IDE, фреймворки, дизайн-инструменты) объединяют ряд ключевых характеристик:

– визуальное проектирование и прототипирование;

– поддержка компонентного подхода (наличие заготовленных компонентов);

– автоматизация и предварительный просмотр;

– интеграция с системами верстки и адаптации под различные размеры экранов.

Рассмотрим существующие среды разработки UI с точки зрения адаптивности интерфейсов, производимых в этих средах.

1.2 Технологии разработки пользовательских интерфейсов

1.2.1 React

React — это декларативная JavaScript-библиотека с открытым исходным кодом, разработанная и поддерживаемая Meta (ранее Facebook), показана на рисунке 1.1. Основное назначение React — создание пользовательских интерфейсов, особенно одностраничных веб-приложений (SPA). React следует компонентному подходу: интерфейс разбивается на независимые, переиспользуемые части (компоненты), каждая из которых управляет собственным состоянием.

React не является полноценным фреймворком (в отличие, например, от Angular), поэтому для маршрутизации, управления состоянием или серверного рендеринга требуются дополнительные библиотеки (React Router, Redux, Zustand, Next.js и др.). Благодаря Virtual DOM React обеспечивает высокую производительность. При обновлении состояния он вычисляет минимальные изменения в DOM и применяет их эффективно.

Сценарии использования:

– веб-приложения с динамическим контентом (соцсети, дашборды, CRM);

– мобильные приложения через React Native;

– сайты с SSR/SSG (с помощью Next.js).

Плюсы:

– высокая производительность благодаря Virtual DOM и оптимизациям (React.memo, useCallback и др.);

– огромная экосистема и сообщество - тысячи библиотек, туториалов, инструментов;

– можно интегрировать в существующие проекты или использовать как основу для новых;

– поддержка TypeScript и современных стандартов JavaScript;

– react native позволяет использовать знания React для мобильной разработки.

Минусы:

– не фреймворк "из коробки". Нужно самостоятельно выбирать решения для маршрутизации, стейт-менеджмента и т.д;

– крутая кривая обучения для новичков: JSX, хуки, контекст, асинхронность, жизненный цикл;

– частые обновления могут требовать рефакторинга кода или миграции зависимостей;

– в чистом SPA без SSR SEO-проблемы (решаются с помощью Next.js или аналогов).

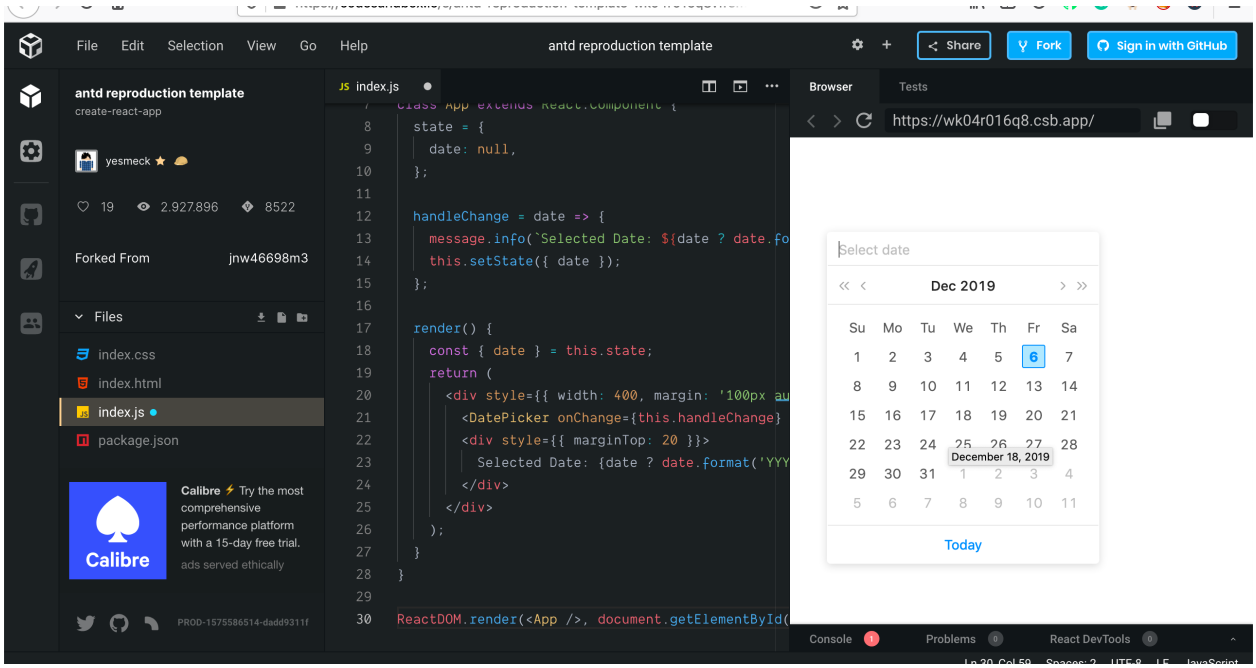

Рисунок 1.1 – Пример использования библиотеки React

1.2.2 Flutter

Flutter — это UI-фреймворк с открытым исходным кодом от Google, предназначенный для кроссплатформенной разработки приложений на iOS, Android, Windows, macOS, Linux и веба с использованием единого кода на языке Dart, показан на рисунке 1.2. В отличие от других решений (например, React Native), Flutter не использует нативные компоненты ОС. Вместо этого он рендерит интерфейс с помощью собственного движка Skia, что обеспечивает одинаковый внешний вид и высокую производительность на всех платформах.

Flutter построен вокруг виджетов — всё в приложении (от кнопки до целого экрана) является виджетом. Это позволяет легко создавать сложные, кастомизированные интерфейсы. Фреймворк активно развивается и поддерживается Google, имеет мощную инструментальную базу (DevTools, Hot Reload) и растущее сообщество.

Сценарии использования:

– ограниченный бюджет и сроки у MVP-продуктов;

– приложения с уникальным дизайном, не привязанным к Material или Cupertino;

– кроссплатформенные корпоративные или потребительские приложения.

Плюсы:

– единая кодовая база для всех платформ — экономия времени и ресурсов;

– высокая производительность (60–120 FPS) благодаря нативной компиляции и собственному рендерингу;

– горячая перезагрузка (Hot Reload) — мгновенное отображение изменений без перезапуска приложения;

– богатая библиотека виджетов и отличная кастомизация;

– активная поддержка Google и быстрое развитие экосистемы.

Минусы:

– размер приложения может быть больше, чем у нативных аналогов;

– язык Dart менее популярен, чем JavaScript или Kotlin/Swift, что может затруднить найм разработчиков;

– ограниченная поддержка некоторых нативных функций без написания платформенного кода (через platform channels);

– меньше готовых решений для нишевых задач по сравнению с нативной разработкой.

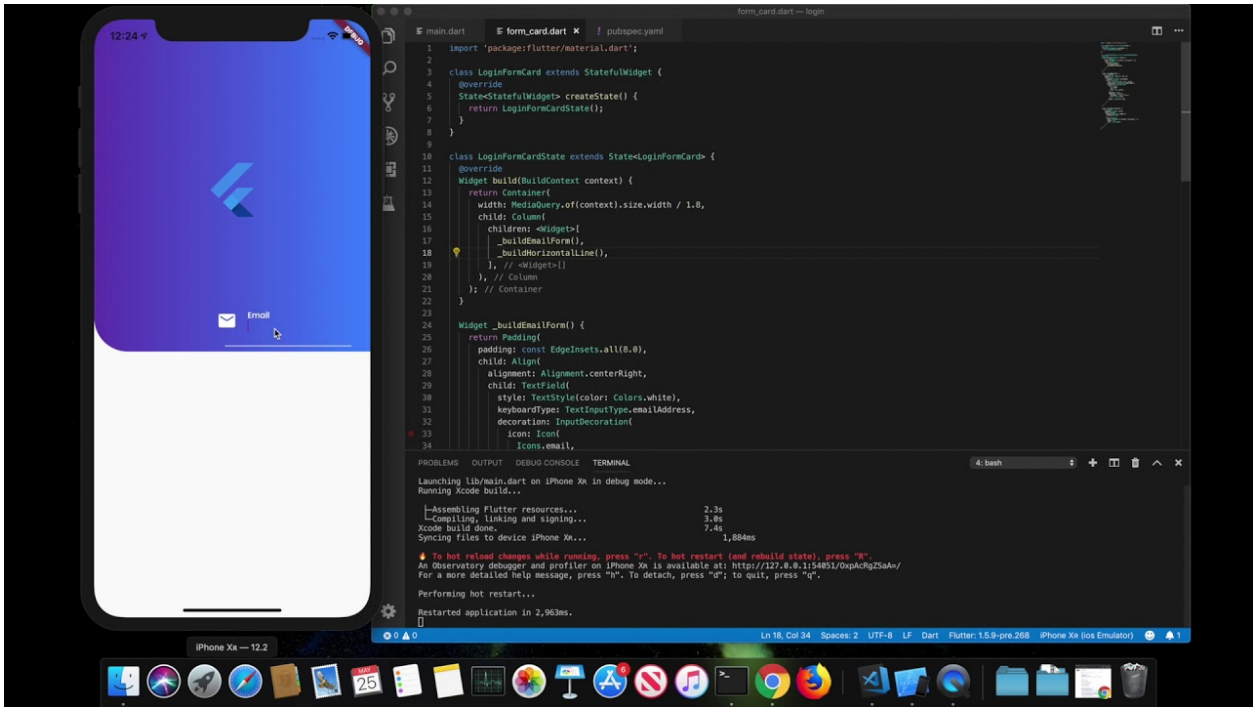

Рисунок 1.2 – Пример использования фреймворка Flutter

1.2.3 Figma

Figma — это облачный инструмент для проектирования интерфейсов, прототипирования и командной работы, который полностью работает в браузере (с опциональным десктопным приложением), показан на рисунке 1.3. Он стал де-факто стандартом в индустрии дизайна благодаря своей простоте, гибкости и мощным возможностям совместной работы.

В Figma можно создавать макеты, компоненты, стили, интерактивные прототипы, а также передавать спецификации разработчикам через Dev

Mode. Инструмент поддерживает плагины, что расширяет его функциональность (автоматизация, экспорт, интеграции с Jira, Zeplin и др.).

Сценарии использования:

– дизайн веб- и мобильных приложений;

– создание дизайн-систем и библиотек компонентов;

– совместная работа дизайнеров, продуктовых менеджеров и разработчиков.

Плюсы:

– работа в реальном времени — несколько человек могут редактировать один файл одновременно;

– кроссплатформенность — доступ из любого браузера, без установки ПО;

– отличная интеграция с разработкой: Dev Mode, экспорт CSS/React/Swift кода, автоматические спецификации;

– бесплатный тариф для индивидуальных пользователей и небольших команд;

– активное сообщество и огромное количество готовых ресурсов (UI-киты, иконки, шаблоны).

Минусы:

– ограниченные возможности для сложной анимации по сравнению с After Effects или Principle;

– зависимость от интернета — офлайн-режим ограничен;

– не предназначен для написания кода — только визуальный дизайн и прототипирование;

– производительность может падать при работе с очень большими файлами.

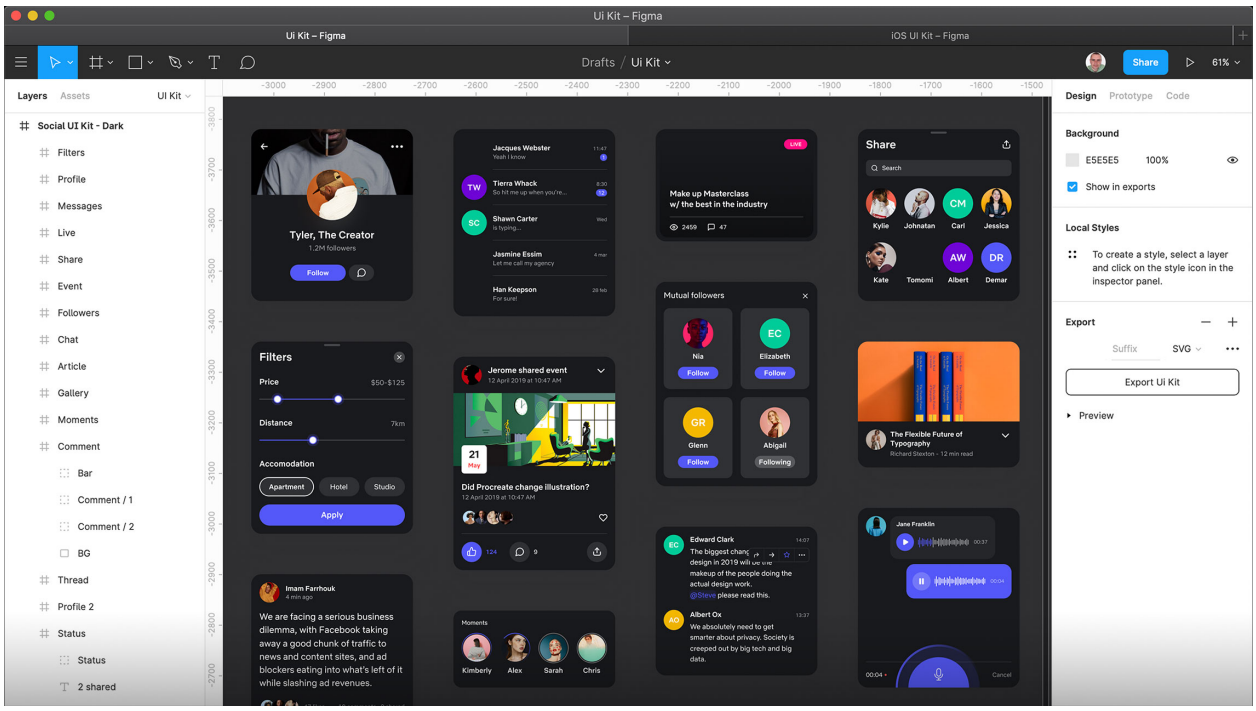

Рисунок 1.3 – Пример использования среды Figma

1.2.4 Material UI (MUI)

Material UI (ныне — MUI) — это самая популярная библиотека React-компонентов, реализующая принципы Material Design от Google, показана на рисунке 1.4. Она предоставляет готовые, адаптивные, доступные и кастомизируемые компоненты: кнопки, формы, таблицы, модальные окна и многое другое.

MUI состоит из нескольких пакетов:

– @mui/material — основные компоненты;

– @mui/system — утилиты для стилизации;

– @mui/icons — иконки Material;

– @mui/lab — экспериментальные компоненты.

Библиотека поддерживает темизацию через CSS-in-JS (Emotion), позволяет легко создавать кастомные темы и интегрируется с TypeScript.

Сценарии использования:

– быстрая разработка внутренних инструментов, админок, дашбордов;

– ограниченный бюджет на дизайн у MVP-продуктов;

– проекты, где важна доступность (a11y) и соответствие стандартам.

Плюсы:

– готовые, протестированные компоненты с поддержкой клавиатуры и скринридеров;

– простая темизация — можно быстро менять цвета, шрифты, отступы;

– отличная документация с живыми примерами и API-справочником;
– активное сообщество и частые обновления;
– поддержка TypeScript "из коробки".

Минусы:

– "шаблонный"внешний вид, если не кастомизировать глубоко;
– большой размер бандла, особенно при импорте всей библиотеки (требуется tree-shaking);
– зависимость от React — нельзя использовать в других фреймворках без оберток;
– сложность глубокой кастомизации у некоторых компонентов (например, Autocomplete).

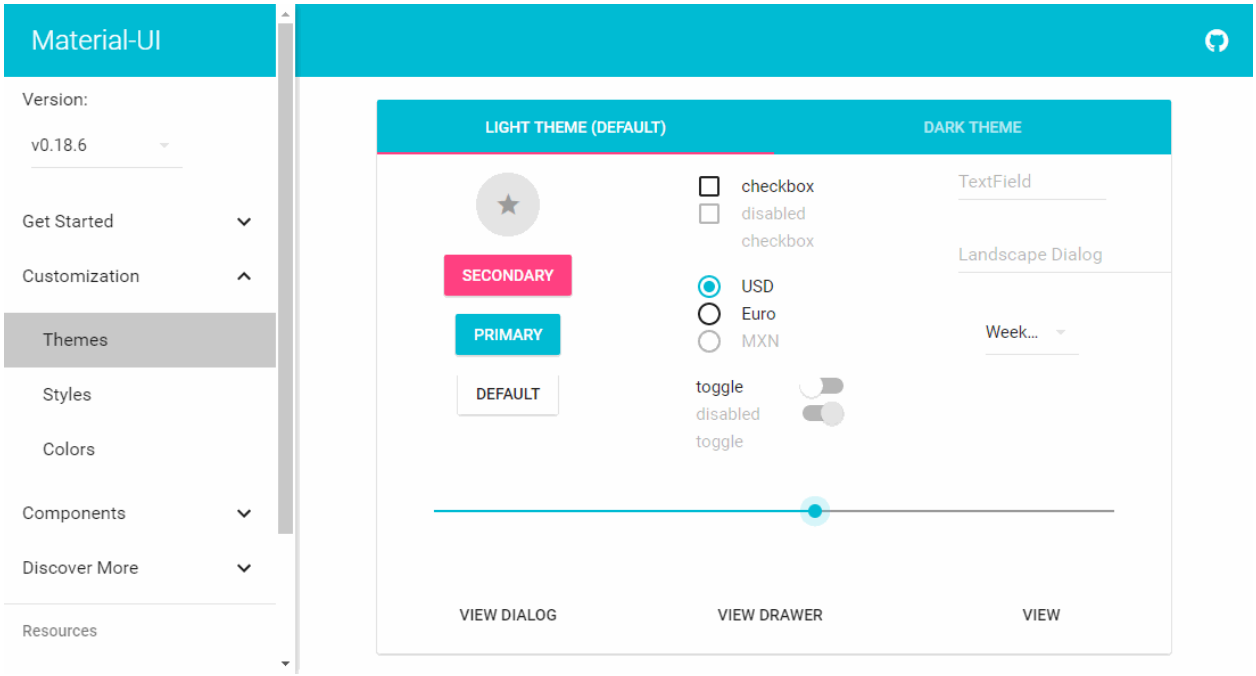

Рисунок 1.4 – Пример использования библиотеки Material UI

1.2.5 Webflow

Webflow — это визуальный конструктор веб-сайтов, сочетающий возможности графического редактора (как Figma) с возможностью генерации чистого HTML, CSS и JavaScript. Он позволяет дизайнерам и нетехническим специалистам создавать адаптивные, интерактивные сайты без написания кода, но с полным контролем над верстkой и стилями, показан на рисунке 1.5.

Webflow включает встроенный хостинг, CMS (для блогов, каталогов, новостей), формулы, анимации и даже базовую логику взаимодействий. На платных тарифах можно экспортировать сгенерированный код.

Сценарии использования:

– лендинги, корпоративные сайты, портфолио.

– прототипирование и быстрая валидация идей.

– сайты с контентом, управляемым через CMS (блоги, каталоги).

Плюсы:

– визуальная разработка без кода, но с семантической и чистой версткой;

– встроенный хостинг и CMS — всё в одной платформе;

– мощные инструменты анимации и интерактива (микровзаимодействия, скролл-анимации);

– адаптивный дизайн с контролем на разных брейкпоинтах;

– экспорт кода (на тарифах Site и выше).

Минусы:

– зависимость от платформы — миграция с Webflow сложна и дорога;

– ограниченная логика — нельзя реализовать сложные веб-приложения (например, с авторизацией, WebSocket, сложными API);

– цена может быть высокой при масштабировании (особенно с CMS и хостингом);

– не подходит для команд разработчиков, предпочитающих контроль через Git и CI/CD.

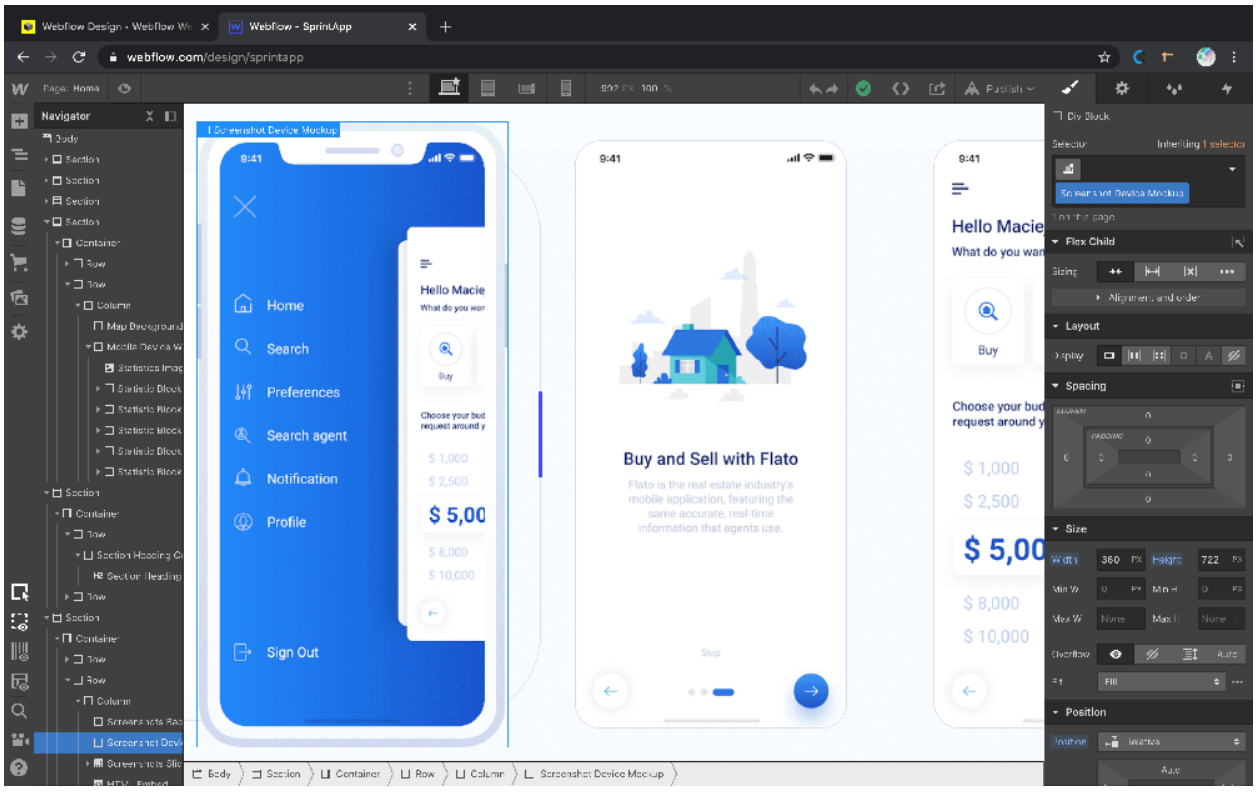

Рисунок 1.5 – Пример использования среды Webflow

1.2.6 Framer

Framer — это современный инструмент для проектирования интерфейсов, прототипирования и даже разработки сайтов, сочетающий визуальный

редактор с возможностью написания кода на React, показан на рисунке 1.6. Изначально Framer был инструментом для продвинутых прототипов с анимациями, но сегодня он превратился в полноценную платформу для создания интерактивных продуктов.

Framer позволяет:

– создавать высокодетализированные прототипы с микровзаимодействиями;

– писать настоящие React-компоненты внутри дизайнера;

– публиковать сайты напрямую с платформы (с доменом, CMS, формами);

– использовать Dev Mode для передачи спецификаций разработчикам.

Сценарии использования:

– продуктовый дизайн с высокой интерактивностью;

– быстрая публикация маркетинговых сайтов и лендингов;

– совместная работа дизайнеров и front-end разработчиков.

Плюсы:

– глубокая интерактивность и анимации — ближе к реальному продукту, чем статичные макеты;

– поддержка React — можно писать и переиспользовать настоящий код;

– современный и интуитивный интерфейс с мощными инструментами (Auto Layout, Variants, Constraints);

– встроенный хостинг и CMS (аналогично Webflow, но с более гибкой логикой);

– передачу проекта разработчикам упрощает Dev Mode.

Минусы:

– платная подписка — бесплатный тариф сильно ограничен;

– меньше готовых ресурсов и сообщества, чем у Figma;

– может быть избыточным для простых задач (например, статичный лендинг);

– зависимость от платформы — как и Webflow, миграция сложна.

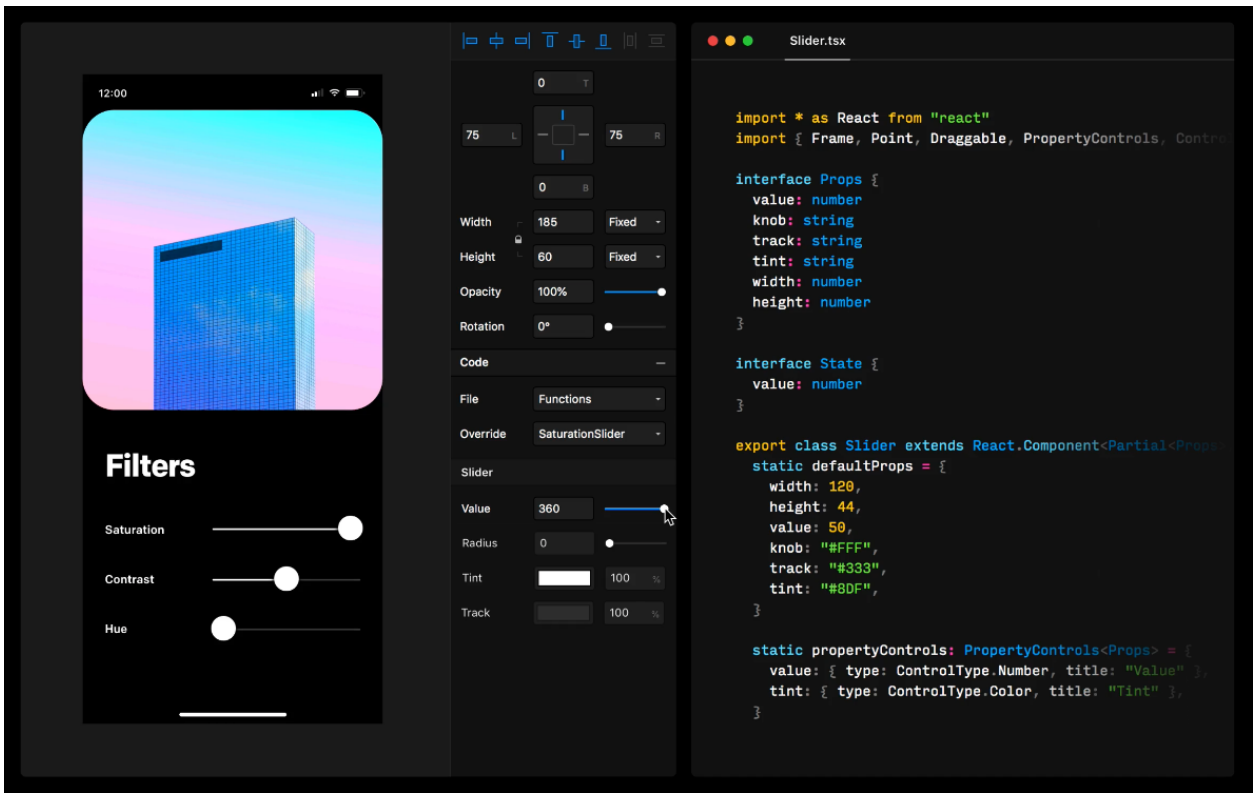

Рисунок 1.6 – Пример использования библиотеки Framer

1.2.7 AdaptForge

AdaptForge — это модельно-ориентированный (MDE) подход для автоматической генерации адаптивных и доступных мобильных интерфейсов, специально разработанный для удовлетворения разнообразных возрастных потребностей пожилых пользователей. Технология была создана исследователями из Monash University (Австралия) и представлена в научной статье 2025 года, показана на рисунке 1.7.

AdaptForge позволяет разработчикам описывать потребности пожилых пользователей (например, снижение зрения, проблемы с моторикой, когнитивные трудности) с помощью двух предметно-ориентированных языков (DSL):

1 Context DSL — моделирует контекст использования: возраст, тип нарушения (зрение, слух, моторика), устройство, окружение (освещённость, шум) и предпочтения пользователя.

2 Adapt DSL — определяет условные правила адаптации интерфейса, например, если пользователь старше 70 лет и имеет низкую контрастную чувствительность, то изменить тему на чёрно-белую и увеличить жирность текста.

Эти модели подаются в MDE-конвейер, который автоматически анализирует исходный код Flutter-приложения (через AST — Abstract Syntax Tree) и генерирует персонализированные версии приложения с адаптированным

UI, не требуя ручного дублирования кода.

AdaptForge поддерживает три типа адаптаций:

– презентационные - изменение размера шрифта, цветовой схемы, контрастности, границ элементов;

– мультимодальные - добавление голосового ввода (speech-to-text) и синтеза речи (text-to-speech);

– навигационные - преобразование сложных форм в пошаговые «мастер-интерфейсы» (wizard-style), чтобы снизить когнитивную нагрузку.

Сценарии использования:

– разработка мобильных приложений для пожилых (e-health, e-banking, ритейл), где важна персонализированная доступность;

– быстрое создание множества адаптированных версий одного приложения под разные группы пользователей без дублирования кода;

– соответствие требованиям стандартов WCAG и законодательства (например, European Accessibility Act);

– поддержка low-code подхода для внедрения доступности в существующие Flutter-приложения с минимальными изменениями (достаточно добавить key к виджетам).

Плюсы:

– автоматизация персонализации — устраняет необходимость вручную создавать отдельные ветки кода для разных групп пользователей;

– глубокая адаптация под возрастные особенности — учитывает не только стандартные настройки ОС (масштаб текста), но и сложные сценарии: сочетание нарушений, окружение, предпочтения;

– интеграция с промышленным стеком (Flutter) — работает с реальными приложениями; для адаптации достаточно добавить уникальные key к виджетам;

– поддержка WCAG и документирование — правила адаптации могут ссылаться на конкретные критерии WCAG; комментарии встраиваются прямо в сгенерированный код;

– положительная оценка разработчиками — в исследовании 16 из 18 Flutter-разработчиков выразили готовность использовать AdaptForge в реальных проектах;

– ориентация на дизайн-системы и персоны — поддерживает моделирование не только индивидуальных пользователей, но и целевых персон (например, пожилая медсестра с низким зрением).

Минусы:

– жёсткая привязка к Flutter — текущая реализация работает только с Flutter/Dart. Порт на React Native, Android или iOS потребует полной переработки DSL и MDE-конвейера;

– сложность сборки и установки — требует настройки Eclipse IDE, EMF, Sirius, Xtext — что непривычно для большинства современных Flutter-

разработчиков, использующих VS Code или Android Studio;

– проблемы масштабируемости и поддержки кода — генерация множества адаптированных версий может привести к взрывному росту кодовой базы, усложнению версионирования и трудностям при отладке;

– зависимость от данных о пользователе — для работы нужны точные данные о возрасте, нарушениях, предпочтениях. Сбор этих данных может вызывать проблемы конфиденциальности и снижать конверсию (сложный онбординг);

– ограничения App Store / Google Play — динамическая подгрузка адаптаций через OTA (over-the-air) может нарушать политики магазинов приложений. Обход (например, через Shorebird) требует дополнительной инфраструктуры;

– только design-time адаптации (пока) — пользователь не может в реальном времени менять настройки — адаптации генерируются до сборки приложения, а не во время его использования.

Хотя технология пока находится на стадии прототипа и имеет ограничения, её подход открывает путь к новому поколению интеллектуальных low-code инструментов, где доступность — не опция, а встроенная часть процесса разработки.

В будущем авторы планируют:

– поддержку run-time адаптаций (в том числе через LLM на основе естественного языка);

– упрощение установки (Eclipse-плагин);

– расширение на другие фреймворки;

– интеграцию с централизованными профилями доступности (единый профиль для всех приложений).

Рисунок 1.7 – Пример использования технологии AdaptForge

1.3 Общие требования к среде разработки пользовательских интерфейсов

1.3.1 Анализ сильных сторон существующих решений

К сильным сторонам исследованных решений можно отнести:
1 Широкий функционал. В каждом из решений довольно широкий

функционал, который позволяет создавать интерфейсы под различные решения.

2 Кроссплатформенность. Существующие решения поддерживают различные платформы и ОС: web, Windows, MacOS, Android, IOS.

3 Легкая интеграция с другими инструментами (интеграция React с TypeScript, поддержка Material UI TypeScript, интеграция решений друг с другом и т. д.).

1.3.2 Анализ недостатков существующих решений

С точки зрения адаптивности производимых интерфейсов, а также наличия обратной связи можно выделить следующие недостатки в средах разработки:

1 Отсутствие обратной связи между средой и интерфейсом (при необходимости внесения изменений в интерфейс сделать это моментально невозможно - требуется взаимодействие пользователя со средой).

2 Недостаток или отсутствие описаний компонентов интерфейса (описание того или иного элемента интерфейса часто бывает коротко или отсутствует).

3 Сложность разработки интерфейсов (несмотря на существующие упрощения процесса разработки порог вхождения в разработку все равно остается высоким).

1.4 Вывод

В ходе анализа существующих подходов к разработке пользовательских интерфейсов и инструментов для их создания стало понятно, что современные технологии уже делают огромный шаг в сторону удобства, кроссплатформенности и персонализации. Такие решения, как React, Flutter, Figma или Webflow, действительно упрощают жизнь разработчиков и дизайнеров — от быстрого прототипирования до генерации готового кода.

Однако, несмотря на всё это разнообразие и мощь, есть ощутимое упущение, что интерфейсы по-прежнему недостаточно "умны". Они не умеют гибко подстраиваться под пользователя в реальном времени, особенно если речь идёт о нетривиальных потребностях — например, у пожилых людей или людей с ограниченными возможностями. Даже продвинутые подходы вроде AdaptForge пока работают только на этапе сборки и сильно привязаны к конкретной технологии (Flutter).

Более того, большинство инструментов не обеспечивают прямой обратной связи между самим интерфейсом и средой разработки, поэтому, если пользователь сталкивается с неудобством, разработчику приходится вручную вносить правки, а не получать автоматические предложения по

адаптации.

Таким образом, несмотря на прогресс, перед разработчиками по-прежнему стоит задача — создать среду, в которой интерфейс не просто статичен или настраивается вручную, а умеет понимать контекст, обучаться и адаптироваться сам, без постоянного вмешательства человека. Это и есть следующий логичный шаг в эволюции UI/UX-разработки.

2 ПРОЕКТИРОВАНИЕ

Основной целью этапа проектирования является разработка архитектуры среды разработки UI для ostis-систем, которая бы устранила выявленные в ходе анализа недостатки, такие как отсутствие динамической обратной связи и сложность описания компонентов.

2.1 Постановка задачи проектирования

На основе проведённого анализа и формальной модели, представленной в научной идее, была сформулирована следующая задача проектирования:

Спроектировать среду разработки пользовательских интерфейсов ostis-систем, состоящую из двух ключевых подсистем:

– база знаний компонентов интерфейса – онтология, содержащая формализованные описания UI-компонентов (кнопок, полей ввода, контейнеров и т.д.), их свойств (цвет, размер, расположение), поведения и связей между ними. Описание должно быть осуществлено на семантическом коде (SCg/SCs) в соответствии с мета-онтологией интерфейсов OSTIS;

– интерпретатор и среда сборки – программный модуль, способный:

– запрашивать у пользователя тексовое описание интерфейса на естественном языке, интерпретировать его, составляя запрос к базе знаний;

– интерпретировать семантические описания компонентов и генерировать на их основе готовый код для UI (например HTML со встроенными CSS и JavaScript);

– обеспечивать механизм для динамической подстройки (адаптации) интерфейса на основе контекста, извлекаемого из базы знаний или поведения пользователя.

Проектирование должно быть выполнено с использованием стека технологий, принятого в экосистеме OSTIS (C++, TypeScript), и обеспечивать возможность последующей программной реализации в рамках курсового проекта.

2.2 Архитектура системы автоматизированного проектирования адаптивных пользовательских интерфейсов

Разрабатываемая система, представленная на рисунке 2.1, представляет собой программный комплекс — конструктор интерфейсов, интегрированный с OSTIS-платформой. Его основное назначение — автоматизация процесса создания адаптивных пользовательских интерфейсов (ПИ) на основе семантических моделей и естественно-языковых описаний, предоставляемых пользователем. Архитектура системы спроектирована с учетом

принципов модульности, масштабируемости и ориентации на повторное использование формализованных компонентов.

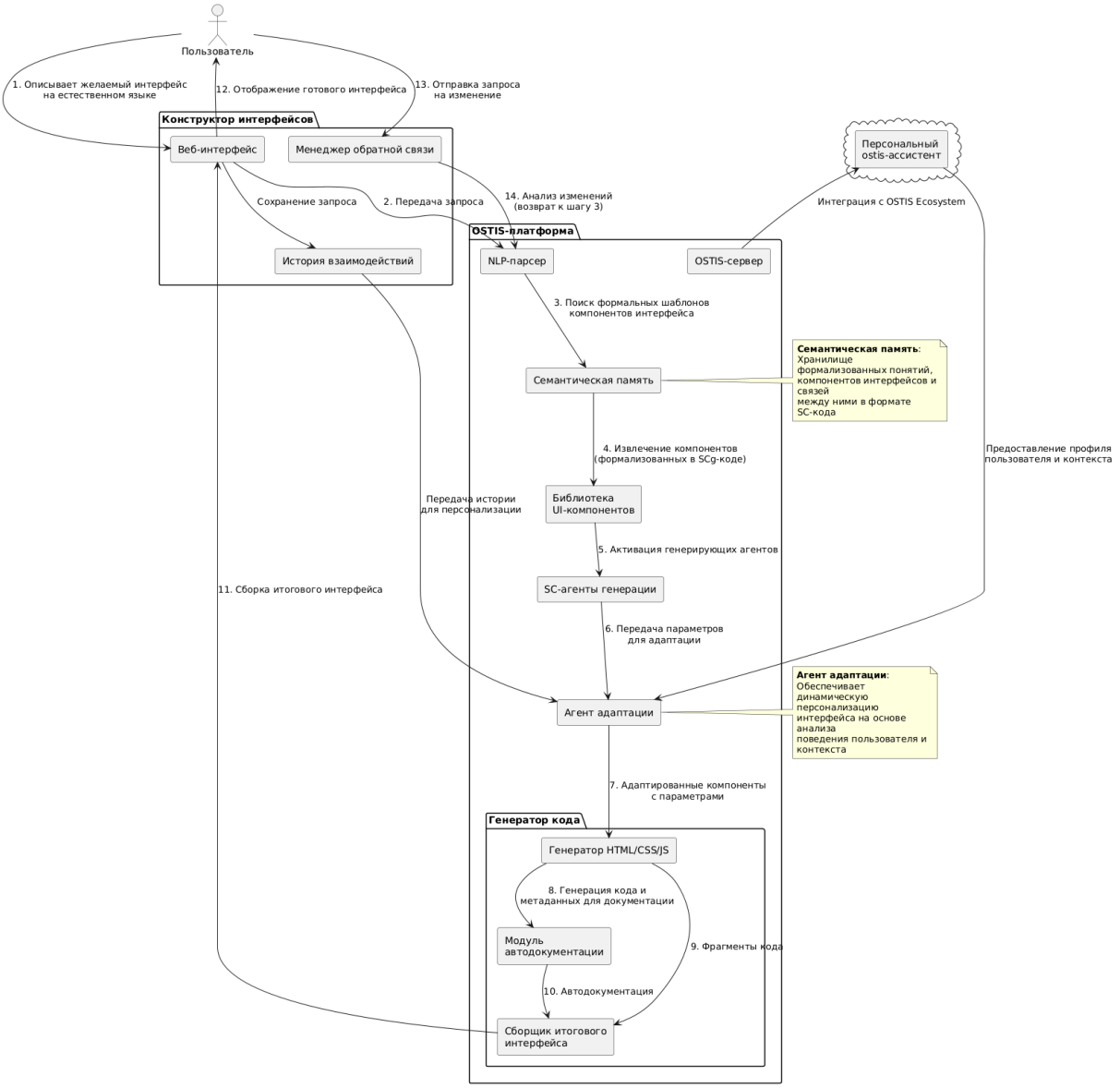

Рисунок 2.1 – Общая схема архитектуры системы

2.2.1 Общая логика работы системы

Процесс взаимодействия с системой можно представить в виде последовательности следующих этапов:

– ввод требований. Пользователь через Веб-интерфейс описывает желаемый интерфейс на естественном языке;

– семантический анализ. NLP-парсер анализирует запрос, выделяя токены, соответствующие компонентам интерфейса и их атрибутам;

– поиск шаблонов. На основе токенов в Семантической памяти (sc-памяти) OSTIS-платформы осуществляется поиск формализованных шаблонов UI-компонентов;

– адаптация и персонализация. Агент адаптации корректирует параметры найденных компонентов на основе истории взаимодействий и профиля пользователя, предоставляемого Персональным ostis-ассистентом;

– генерация кода. SC-агенты активируют Генератор HTML/CSS/JS, который на основе адаптированных шаблонов производит код отдельных компонентов;

– формирование документации. Модуль автодокументации генерирует описание интерфейса на основе семантических описаний компонентов, хранящихся в БЗ;

– сборка интерфейса. Сборщик объединяет сгенерированные фрагменты кода и документацию в единый веб-интерфейс;

– предоставление результата. Готовый интерфейс отображается пользователю через Веб-интерфейс;

– обратная связь и итерация. Пользователь может отправить запрос на изменение через Менеджер обратной связи, после чего цикл обработки повторяется, начиная с этапа семантического анализа.

2.2.2 Ключевые компоненты архитектуры

Архитектура системы разделена на два крупных пакета: «Конструктор интерфейсов» и «OSTIS-платформа».

1 Пакет "Конструктор интерфейсов":

Данный пакет реализует уровень взаимодействия с конечным пользователем.

– веб-интерфейс (Web UI). Является точкой входа в систему. Предоставляет пользователю возможность ввода текстового описания интерфейса и визуализации готового результата. Реализован с использованием современных веб-технологий (HTML, CSS, JavaScript/TypeScript).

– менеджер обратной связи (Feedback Manager). Фиксирует запросы пользователя на модификацию интерфейса и передает их в NLP-парсер для повторного анализа, замыкая thus цикл обратной связи.

– история взаимодействий (Interaction History). Осуществляет логирование всех запросов и действий пользователя. Накопленные данные передаются Агенту адаптации для персонализации будущих интерфейсов.

2 Пакет "OSTIS-платформа":

Это ядро системы, отвечающее за хранение знаний, их обработку и генерацию кода.

– NLP-парсер. Осуществляет синтаксический и семантический разбор естественно-языкового запроса. На основе заранее определенных шаблонов и закономерностей производит токенизацию, выделяя сущности (например,

«кнопка», «поле ввода») и их атрибуты («красный», «в центре»);

– семантическая память (sc-memory). Централизованное хранилище знаний, реализованное в виде графовой базы данных. Содержит формализованные в SC-коде понятия, включая онтологию компонентов пользовательского интерфейса, их свойства, поведение и взаимосвязи;

– библиотека UI-компонентов. Коллекция переиспользуемых, формализованных описаний стандартных элементов интерфейса (кнопок, полей, меню и т.д.), хранящаяся в sc-памяти;

– SC-агенты генерации. Программные агенты, работающие с SC-кодом. Активируются после успешного поиска шаблонов и отвечают за инициирование процессов генерации и адаптации;

– агент адаптации (Adaptation Agent). Ключевой компонент для обеспечения персонализации. На основе данных из Истории взаимодействий и профиля от Персонального ostis-ассистента модифицирует параметры найденных UI-компонентов (например, изменяет цветовую схему, размеры шрифтов, компоновку элементов) для соответствия индивидуальным предпочтениям и контексту пользователя.

3 Подпакет "Генератор кода":

Данный подпакет отвечает за трансляцию семантических моделей в исполняемый код.

– генератор HTML/CSS/JS. На основе адаптированных формальных шаблонов из sc-памяти генерирует валидный код фронтенда (HTML-разметку, CSS-стилизацию и JavaScript-логику) для каждого компонента интерфейса;

– модуль автодокументации. Извлекает из семантических описаний компонентов текстовые метаданные (назначение, параметры, связи) и формирует встроенную документацию, которая «упаковывается» в итоговый код (например, в виде комментариев или отдельного справочного раздела);

– сборщик итогового интерфейса (Interface Builder). Агрегирует сгенерированные фрагменты кода и документацию, разрешает зависимости между компонентами и формирует единый, готовый к использованию веб-файл (например, ‘index.html‘).

4 Внешняя интеграция:

Система тесно интегрирована с экосистемой OSTIS через облачный компонент.

– персональный ostis-ассистент - внешний интеллектуальный агент, который предоставляет контекстную информацию о пользователе (его профиль, роли, типичные задачи) Агенту адаптации, что позволяет осуществлять глубокую персонализацию генерируемых интерфейсов.

2.2.3 Научно-техническое обоснование выбранной архитектуры

Предложенная архитектура является практической реализацией выдвинутой научной гипотезы о возможности использования единой семантической модели для автоматизированного синтеза и адаптации ПИ.

– семантическая модель как единый источник истины. Все компоненты и их свойства формализованы в SC-коде и хранятся в sc-памяти, что обеспечивает согласованность данных на всех этапах обработки — от анализа запроса до генерации кода.

– агентно-ориентированный подход. Использование SC-агентов и Агента адаптации позволяет создать гибкую, событийно-управляемую систему, способную к динамической перестройке процессов в реальном времени.

– компонентный подход и повторное использование. Библиотека формализованных UI-компонентов обеспечивает стандартизацию и многократное использование элементов, что снижает трудозатраты на разработку и повышает надежность итоговых интерфейсов.

– поддержка полного жизненного цикла. Архитектура охватывает не только создание, но и последующую модификацию интерфейса на основе обратной связи, реализуя итеративный процесс разработки, управляемый пользователем.

2.3 Пользователи системы

В соответствии с use case диаграммой, представленной на рисунке 2.2, система взаимодействует с тремя основными категориями пользователей, каждая из которых имеет определенные цели и задачи при работе с конструктором адаптивных интерфейсов.

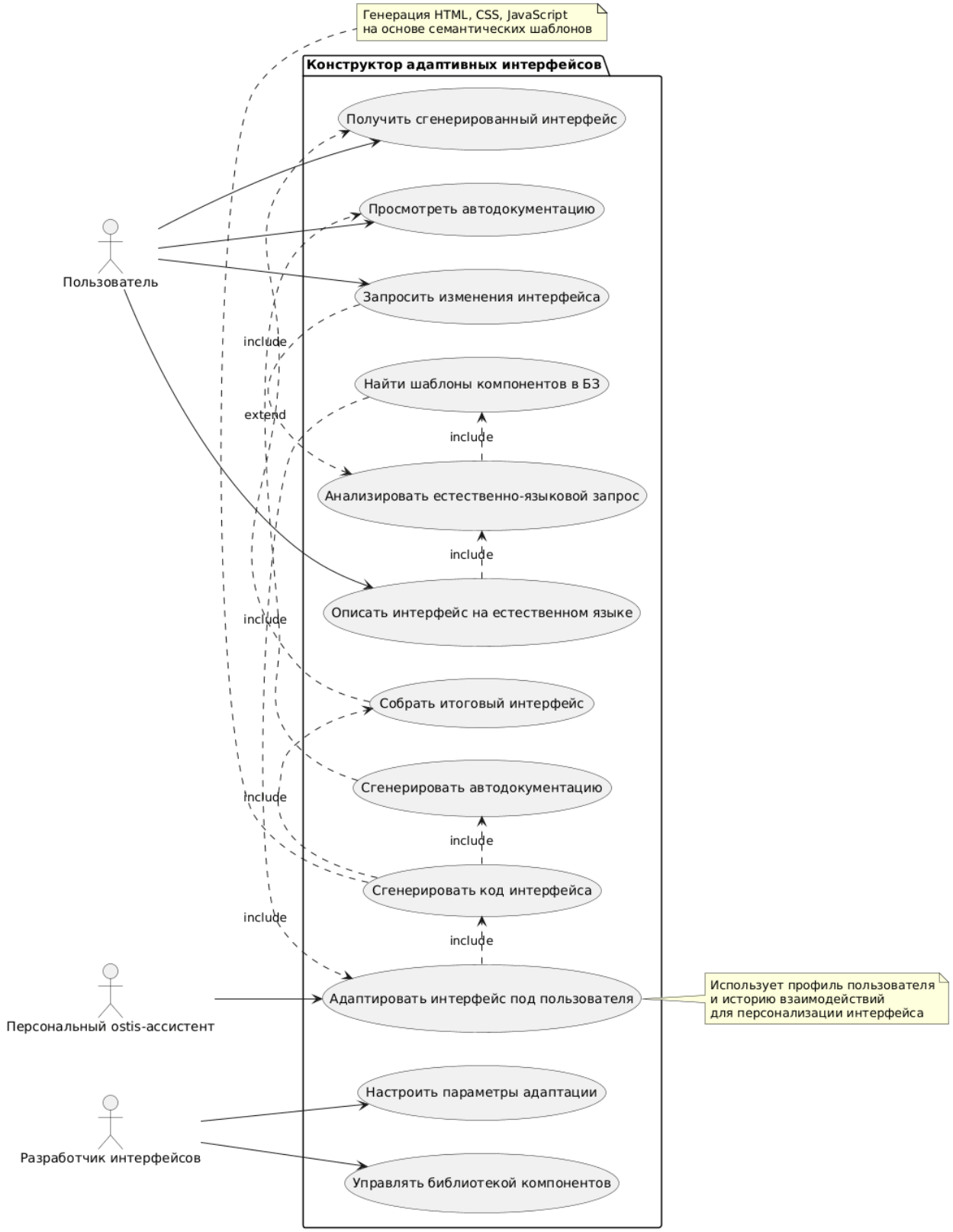

Рисунок 2.2 – Сценарии использования

Таблица 2.1 – Категории пользователей системы

Категория пользователя	**Описание и основные сценарии использования**
Пользователь	Конечный пользователь системы, который заинтересован в быстром получении адаптивного пользовательского интерфейса без необходимости написания кода. Основные сценарии: – описание желаемого интерфейса на естественном языке; – получение сгенерированного интерфейса в формате HTML/CSS/JS; – запрос изменений и модификаций интерфейса через систему обратной связи; – просмотр автоматически сгенерированной документации к интерфейсу
Персональный ostis-ассистент	Интеллектуальный агент, интегрированный в экосистему OSTIS, который предоставляет контекстную информацию о пользователе для персонализации интерфейсов. Основные сценарии: – предоставление профиля пользователя и его предпочтений; – передача контекстной информации о текущих задачах пользователя; – обеспечение динамической адаптации интерфейса на основе анализа. поведения
Разработчик интерфейсов	Специалист, отвечающий за развитие и поддержку системы конструктора интерфейсов. Основные сценарии: – управление библиотекой семантически совместимых UI-компонентов; – настройка параметров и правил адаптации интерфейсов; – расширение функциональности системы и добавление новых компонентов; – мониторинг и анализ эффективности работы системы.

2.3.1 Взаимодействие между категориями пользователей

Взаимодействие между различными категориями пользователей образует замкнутый цикл разработки и совершенствования интерфейсов:

– пользователь инициирует процесс создания интерфейса через естественно-языковое описание и получает готовый результат;

– персональный ostis-ассистент обеспечивает контекстную адаптацию интерфейса под индивидуальные потребности пользователя;

– разработчик поддерживает и развивает систему, обновляя библиотеку компонентов и совершенствуя алгоритмы адаптации.

2.4 Предлагаемый путь решения поставленной задачи проектирования

Ввиду высокой сложности поставленной задачи проектирования было принято решение о разбиении проекта на отдельные этапы и распределении областей ответственности между участниками проекта.

В частности в данной пояснительной записке рассматривается проблема формализации наиболее распространенных компонентов интерфейса.

Вышеупомянутые компоненты можно разделить на следующие категории:

1 Компоненты вывода:

– изображение;

– видео;

– звук;

– текст.

2 Декоративные элементы:

– разделитель;

– пустое пространство.

3 Контейнеры:

– меню + строка меню;

– списковый контейнер + строка спискового контейнера;

– панель прокрутки;

– окно.

4 Интерактивные элементы (ввод данных):

– многострочное текстовое поле;

– однострочное текстовое поле;

– ползунок;

– для нескольких значений;

– для одного значения;

– флаговая кнопка;

– радиокнопка;

– переключатель;
– выбираемый элемент.
5 Компонент запроса действий:
– кнопка;
– пункт меню.

В рамках практической части реализации проекта особое внимание будет уделено конкретным элементам интерфейса, представляющим наибольший интерес с точки зрения формализации и практического применения. В частности, детальному рассмотрению будут подвергнуты следующие компоненты:

– кнопка (button) — как ключевой элемент взаимодействия с системой и выполнение/вызов определенной функции.

2.5 Вывод

1 Постановка задачи. Спроектирована архитектура среды разработки UI для ostis-систем, состоящая из базы знаний компонентов и интерпретатора с генератором кода.

2 Архитектура системы. Разработана модульная архитектура с четким разделением на конструктор интерфейсов и OSTIS-платформу, поддерживающая полный цикл от NLP-анализа до генерации адаптивного кода.

3 Пользователи системы. Определены три категории пользователей (конечный пользователь, персональный ассистент, разработчик) с конкретными сценариями взаимодействия.

4 Путь решения. Для первоочередной реализации выбран ключевой компонент интерфейса - кнопка (button).

3 РАЗРАБОТКА И ФОРМАЛИЗАЦИЯ КОМПОНЕНТА

3.1 Создание прототипа интерфейса

Для решения задачи формализации необходимо полное понимание принципа построения html документа, представляющего интерфейс. Также требуется понимать, как устроен каждый компонент, и как он взаимосвязан с другими элементами.

3.1.1 Графическое изображение интерфейса

Был реализован прототип интерфейса, показанный на рисунке 3.1, включающий в себя основные компоненты и связи между ними.

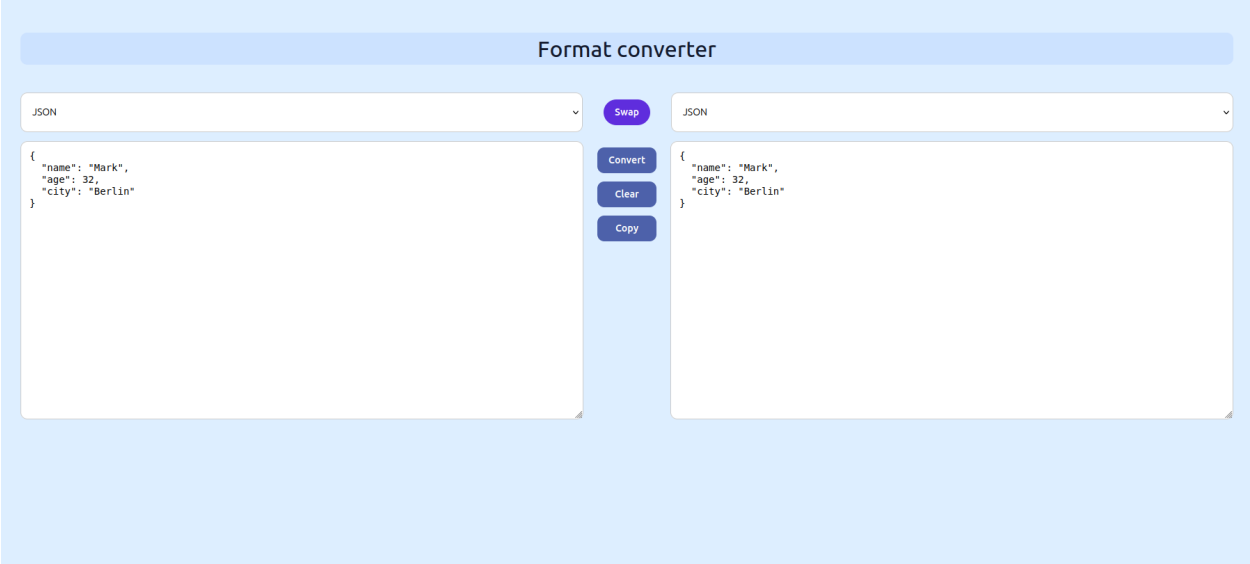

Рисунок 3.1 – Прототип интерфейса

3.1.2 Пример кода некоторых элементов интерфейса

– на листинге ниже показан пример кода выпадающего меню:

```
<select
    id="fromFormat"
    class="format-select"
    style="
        flex: 1;
        padding: 10px 16px;
        font-size: 14px;
        border: 1px solid #cccccc;
        border-radius: 10px;
```

```
        background: #ffffff;
    "
>
 <option value="json">JSON</option>
    <option value="xml">XML</option>
    <option value="yaml">YAML</option>
</select>
```

– на листинге ниже показан пример кода многострочного поля ввода текста:

```
<textarea
 id="inputData"
 placeholder="Input data..."
 style="
  width: 100%;
  height: 400px;
  padding: 12px;
  font-family: monospace;
  font-size: 14px;
  border: 1px solid #cccccc;
  border-radius: 10px;
  background: #ffffff;
  resize: vertical;
 "
>
 {
  "name": "Mark",
  "age": 32,
  "city": "Berlin"
 }
</textarea>
```

– на листинге ниже показан пример кода кнопки и её скрипта:

```
<button
 id="clearBtn"
 class="action-btn"
 style="
  padding: 10px 16px;
  font-size: 14px;
  border: none;
  border-radius: 10px;
```

```
  background: #4d61aa;
  color: white;
  font-weight: bold;
  cursor: pointer;
 "
>
 Clear
 <script>
  function clearAll() {
   document.getElementById("inputData").value = "";
   document.getElementById("outputData").value = "";
  }
 </script>
</button>
```

3.2 Формализация компонентов

В данном подразделе показана конкретная реализация компонента и его формализованная версия на языке SCg.

3.2.1 Компонент Кнопка (button)

На листинге ниже показан пример кода элемента на HTML:

```
<button
    id="copyBtn"
    class="action-btn"
    style="
        padding: 10px 16px;
        font-size: 14px;
        border: none;
        border-radius: 10px;
        background: #4d61aa;
        color: white;
        font-weight: bold;
        cursor: pointer;
    "
>
    Copy
</button>
```

Формализация на SCg рисунке 3.2 и рисунке 3.3:

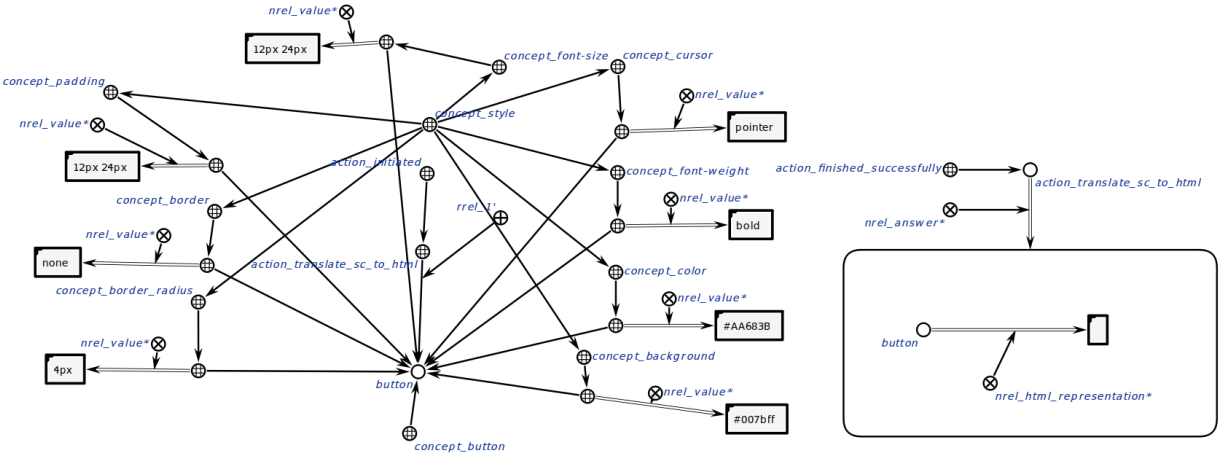

Рисунок 3.2 – Формализация компонента "кнопка"(1)

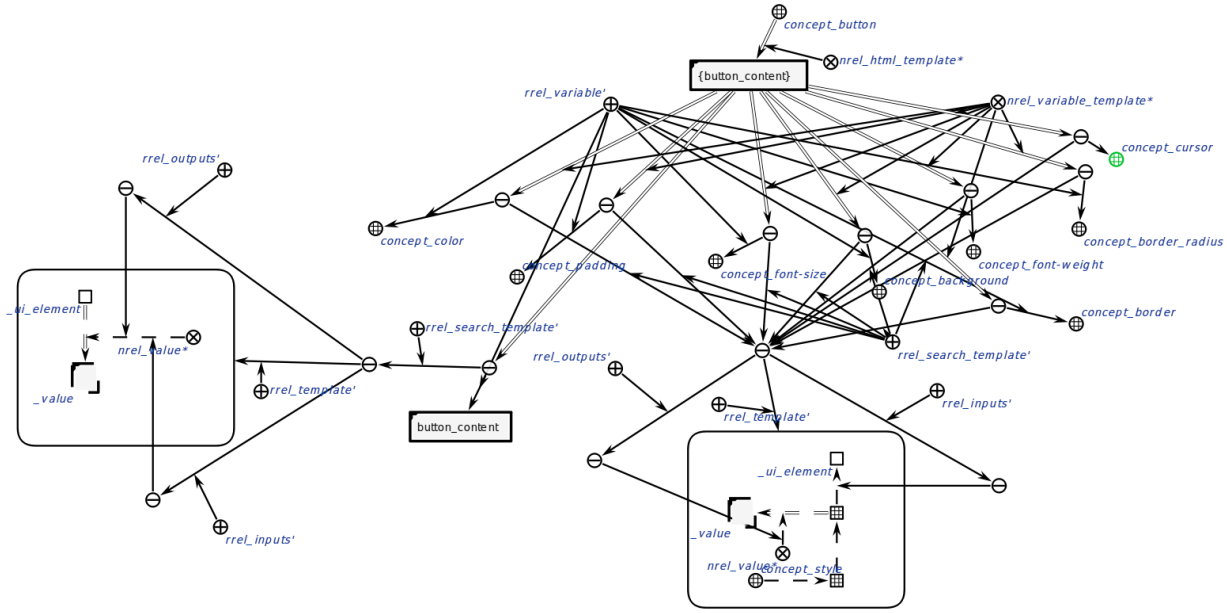

Рисунок 3.3 – Формализация компонента "кнопка"(2)

3.3 Вывод

1 Создан прототип интерфейса, демонстрирующий основные компоненты и их взаимосвязи, что позволило получить полное представление о структуре HTML-документа и принципах построения пользовательского интерфейса.

2 Проведен анализ примеров кода ключевых элементов интерфейса (выпадающее меню, многострочное поле ввода, кнопка со скриптом), что позволило выявить их структуру, атрибуты и стилевые характеристики.

3 Выполнена формализация ключевого компонента интерфейса на языке SCg:

– кнопка (button) — определены основные стилевые атрибуты (цвет, граница, выравнивание, отступы), а так же связь с компонентом "текстовое поле";

4 Разработанная SCg-модель компонента представляет собой семантическое описание, которое может быть использовано в базе знаний системы для последующей генерации кода и адаптации интерфейсов.

ЗАКЛЮЧЕНИЕ

В рамках курсового проекта была выполнена следующая работа:

1 Проведен анализ современных технологий разработки пользовательских интерфейсов (React, Flutter, Figma, Material UI, Webflow, Framer, AdaptForge) и выявлены их ключевые преимущества и недостатки с точки зрения адаптивности и обратной связи.

2 Спроектирована архитектура среды разработки UI для ostis-систем, включающая базу знаний компонентов интерфейса и интерпретатор с генератором кода. Архитектура поддерживает полный жизненный цикл разработки - от NLP-анализа естественно-языковых описаний до генерации адаптивного HTML/CSS/JS кода.

3 Определены три категории пользователей системы (конечный пользователь, персональный ostis-ассистент, разработчик интерфейсов) и их сценарии взаимодействия.

4 Разработан прототип интерфейса и проведен анализ структуры HTML-компонентов, что позволило понять принципы построения пользовательских интерфейсов и взаимосвязи между элементами.

5 Выполнена формализация ключевого компонента интерфейса (кнопка) на языке SCg, создано его семантическое описание для базы знаний системы.

В результате реализации проекта создана основа для семантически-ориентированной среды разработки адаптивных пользовательских интерфейсов, устраняющей выявленные в анализе недостатки существующих решений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Садовский, М. Е. Индивидуализация пользовательских интерфейсов интеллектуальных систем на основе семантической модели / М. Е. Садовский // Цифровая трансформация. — 2023. — Т. 29, № 3. — с. 54–63.

[2] Садовский, М. Е. Семантические модели и средства проектирования адаптивных пользовательских интерфейсов интеллектуальных систем / М. Е. Садовский // Информатика. — 2023. — Т. 20, № 3. — с. 74–89.

[3] Zotov, N. Design principles, structure, and development prospects of the software platform of ostis-systems = Принципы проектирования, структура и перспективы развития программной платформы ostis-систем / N. Zotov // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS) : сборник научных трудов / Белорусский государственный университет информатики и радиоэлектроники ; редкол.: В. В. Голенков [и др.]. – Минск, 2023. – Вып. 7. – С. 67–76.

[4] Sadouski, M. E. Ontological approach to the building of semantic models of user interfaces / M. E. Sadouski // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2021) : сборник научных трудов / Белорусский государственный университет информатики и радиоэлектроники ; редкол.: В. В. Голенков [и др.]. – Минск, 2021. – Вып. 5. – С. 105–116.

[5] Грибова, В. В. Автоматизация разработки программных интерфейсов / В. В. Грибова // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2011): материалы Междунар. научн.-техн. конф., Минск, 10–12 февраля 2011 г. / Бел. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков [и др.]. – Мн., 2011. – С. 287.

[6] Корончик, В. В. Компонентное проектирование программных интерфейсов / В. В. Корончик // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2012): материалы II Междунар. научн.-техн. конф., Минск, 16–18 февраля 2012 г. / Бел. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков [и др.]. – Мн., 2012. – С. 339.

[7] Корончик, В. В. Программный интерфейс IMS OSTIS / В. В. Корончик // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2014): материалы IV междунар. науч.-техн. конф., Минск, 20–22

февраля 2014 г. / Бел. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков [и др.]. – Мн., 2014. – С. 79.

[8] Корончик, В. В. Программный интерфейс IMS OSTIS: новый подход / В. В. Корончик // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2015): материалы V междунар. науч.-техн. конф., Минск, 19–21 февраля 2015 г. / Бел. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков [и др.]. – Мн., 2015. – С. 89.

[9] Шункевич, Д. В. Классы команд интерфейса OSTIS / Д. В. Шункевич // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2016): материалы VI междунар. науч.-техн. конф., Минск, 18–20 февраля 2016 г. / Бел. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков [и др.]. – Мн., 2016. – С. 131.

[10] Сытник, А. А. Онтология предметной области «Удобство использования программного обеспечения» / А. А. Сытник, Т. Э. Шульга, Н. А. Данилов // Труды ИСП РАН. – 2018. – Т. 30, № 2. – С. 195–214.

[11] Данилов, Н. А. Method for constructing a heat map based on the point data of the application user's activity / Н. А. Данилов, Т. Э. Шульга // Прикладная информатика = Journal of Applied Informatics. – 2015. – Т. 10, № 2 (56). – С. 49–58.

[12] Tourwé, T. Ontology-driven Elicitation of Multimodal User Interface Design Recommendations / T. Tourwé, E. Tsiporkova, N. González-Deleito, A. Hristoskova // Journal of Applied Informatics. – 2016. – Vol. 11, № 4. – P. 513–539.

[13] Silva, Thiago Rocha A Formal Ontology for Describing Interactive Behaviors and Supporting Automated Testing on User Interfaces / Thiago Rocha Silva, Jean-Luc Hak, Marco Winckler // International Journal of Semantic Computing. – 2017. – Vol. 11, № 4. – P. 513–539.

[14] Towards an Ontology-based Approach to Develop Software Systems with Adaptive User Interface : [Electronic resource] / Alexandre A. C. de Freitas, Simone D. Costa, Murilo B. Scalser, Monalessa P. Barcellos [et al.]. – 2022. – URL: https://www.researchgate.net/publication/364609123_Towards_an_Ontology-based_Approach_to_Develop_Software_Systems_with_Adaptive_User_Interface (date of access: 01.10.2025).