РОЛЬ И МЕСТО TKINTER В СОВРЕМЕННЫХ РЕАЛИЯХ СОЗДАНИЯ КЛИЕНТСКОЙ ЧАСТИ ТЕХНОЛОГИЧЕСКИХ ПРОДУКТОВ

Гурина В. М.

ФГБОУ ВО «Мелитопольский государственный университет», Мелитополь, e-mail: skald.beregovoi@yandex.ru

Научный руководитель: Береговой А.В.

Введение

Tkinter это одна из популярных библиотек Python. Слово «tkinter» происходит от фразы «Tk interface». Tkinter – это объектно-ориентированная надстройка языка Python над Tcl/Tk – где Tcl (Tool Command Language) является скриптовым языком, а Tk – библиотекой графических виджетов для этого языка. Его можно представить как переводчик с Python на язык Tcl. Мы пишем программу на Python, а код модуля Tkinter переводит ваши инструкции на язык Tcl, который понимает библиотека Tk [1].

Tcl здесь играет роль универсального исполнителя команд – это интерпретируемый язык, специально созданный для встраивания в приложения. Он работает как «движок» для Tk, выполняя низкоуровневые операции по созданию и управлению графическими элементами интерфейса. Вся суть кроссплатформенности возможна именно благодаря тому, что Tcl/Tk существует отдельно от Python и уже решал задачи создания GUI на разных операционных системах.

Цель исследования: анализ библиотеки Tkinter и обоснование актуальности её применения в изучении программирования и в профессиональной деятельности для создания приложений и прототипирования интерфейсов.

Материал и методы исследования

В статье использовано сравнение современных фреймворков для создания приложений на языке программирования Python. Было написано вэб-приложение, что показывает сочетание эффективности библиотеки и простоты её изучения и применения в реальных проектах.

Результаты исследования и их обсуждение

Важнейшей характеристикой Tkinter является его статус как стандартного компонента экосистемы Python. Данный модуль включается во все официальные дистрибутивы языка, что обеспечивает его доступность без необходимости дополнительной установки зависимостей – в том числе потому, что Tcl/Tk обычно предустановлен на большинстве операционных систем. Эта особенность делает Tkinter универсальным решением, гарантирующим работоспособность графических интерфейсов на различных операционных платформах, включая Windows, Linux и macOS.

Фундаментальным аспектом работы с Tkinter является его событийно-ориентированная модель. Библиотека функционирует на основе главного цикла обработки событий (main loop), который непрерывно отслеживает пользовательские действия (щелчки мыши, нажатия клавиш, перемещения курсора) и системные сообщения. В ответ на эти события вызываются соответствующие обработчики – callback-функции, предоставленные разработчиком.

Синтаксически работа с Tkinter характеризуется минималистичным подходом. Создание базового графического интерфейса требует написания всего нескольких строк кода, что делает библиотеку особенно доступной для начинающих разработчиков. Однако за этой кажущейся простотой скрывается достаточно мощная и гибкая система, позволяющая создавать сложные интерфейсные решения через комбинацию базовых компонентов и кастомных настроек. Следует отметить, что несмотря на свою универсальность, Tkinter демонстрирует определенную архитектурную консервативность. Библиотека сохраняет обратную совместимость с ранними версиями Python, что обеспечивает стабильность существующих приложений, но одновременно ограничивает внедрение современных паттернов проектирования пользовательских интерфейсов.

Tkinter, за своей репутацией простого инструмента для новичков, скрывает ряд интересных и мощных возможностей, которые позволяют выходить далеко за рамки создания стандартных форм с кнопками. Одной из таких является виджет Canvas (холст). Это не просто область для рисования линий, а целая интерактивная среда, на которой можно размещать графические примитивы, изображения, текст и даже другие окна. С его помощью можно создавать не только диаграммы и графики, но и полноценные анимированные модели, простые 2D-игры или специализированные редакторы, где объекты можно перемещать, масштабировать и удалять с помощью мыши. Например, вы можете визуализировать алгоритмы сортировки, где элементы будут переставляться с анимацией, или создать интерфейс для управления моделью умного дома, где устройства представлены в виде иконок на плане [2].

Другая часто упускаемая из виду возможность – это расширенная система событий (bindings). В Tkinter можно привязать обработчик не только к клику мыши, но и к наведению курсора, нажатию конкретных комбинаций клавиш, изменению размера окна и даже к собственным, сгенерированным программистом событиям. Это открывает путь к созданию чрезвычайно отзывчивых интерфейсов. Не стоит забывать и о модуле ttk, который представляет собой современную версию классических виджетов. Он включает

такие сложные компоненты, как TreeView – идеальный инструмент для отображения табличных или иерархических данных, например, списка файлов с атрибутами или структуры базы данных. Виджет Notebook позволяет организовать интерфейс с вкладками, что критически важно для сложных приложений, где нужно группировать функциональность. А Progressbar и Combobox делают интерфейс интуитивно понятным и профессиональным [3]. А так же гибкая система стилей и тем в ttk позволяет уйти от устаревшего внешнего вида «по умолчанию». Вы можете не просто менять цвета и шрифты, а переопределять внешний вид всех элементов управления, создавая уникальный и современный дизайн, который будет соответствовать бренду или просто радовать глаз. В совокупности эти возможности превращают Tkinter из инструмента для создания прототипов в платформу для построения полноценных, функциональных и визуально приятных десктопных приложений, которые идеально подходят для внутренних инструментов, утилит сопровождения или научных визуализаций, где важна быстрая разработка и минимальные зависимости.

Приложение Tkinter обычно имеет иерархическую структуру [4]:

- Создать корневое окно с помощью конструктора Tk().
- Добавить в корневое окно виджеты, такие как кнопки, надписи и поля ввода.
- Упорядочить виджеты с помощью менеджеров компоновки, таких как pack(), grid() или place().
- Привязывать события к виджетам, чтобы управлять взаимодействием с пользователем.

Рассмотрим простой пример Tkinter, который создает окно с надписью (рисунок):

```
   import tkinter as tk
def main():
   # Создание главного окна приложения
   root = tk.Tk()
   root.title(«Мое первое Tkinter приложение»)
   root.geometry(«300x100»)

   # Создание текстовой метки
   label = tk.Label(
      root,
      text=»Hello, Tkinter!»,
      font=(«Arial», 14),
      fg=»blue»,
      bg=»lightyellow»
   )
   label.pack(pady=20)

   # Запуск главного цикла обработки событий
   root.mainloop()

# Запуск главной функции
if __name__ == «__main__»:
   main()
```

Простое приложение с библиотекой Tkinter

Несмотря на то, что Tkinter традиционно позиционируется как инструмент для десктопных приложений, он обладает комплексом возможностей, релевантных для задач фронтенд-разработки в широком понимании этого термина [5]:

1) Tkinter имплементирует компонентный подход, где каждый элемент интерфейса (виджет) представляет собой самостоятельную сущность с инкапсулированным состоянием и поведением. Это напрямую коррелирует с компонентной моделью React или Vue.js. Например, виджет Button в Tkinter:

- Обладает свойствами (text, state, command)
- Имеет собственный жизненный цикл (создание, конфигурация, уничтожение)
- Поддерживает систему событий (bindings)

2) Tkinter предоставляет механизмы для управления состоянием, аналогичные state management в веб-фреймворках:

- Переменные Tkinter (StringVar, IntVar, BooleanVar) как реактивные источники данных
- Автоматическое обновление интерфейса при изменении переменных
- Возможность создания унифицированного хранилища состояния

3) Модуль ttk предлагает систему стилей, концептуально близкую к CSS:

```
style = ttk.Style()
style.configure('Custom.TButton',
   padding=[10, 5],
   font=('Arial', 12),
   background='#3498db')
```

Данный подход позволяет реализовать:

- Централизованное управление внешним видом
- Каскадное применение стилей
- Поддержку различных тем оформления

4) Tkinter поддерживает принцип композиции через:

- Создание кастомных виджетов на основе существующих
- Наследование и расширение функциональности
- Модульную организацию интерфейсных компонентов

5) Tkinter естественным образом поддерживает паттерны, характерные для фронтенд-разработки:

- Model-View-Controller через разделение данных и представления

- Observer для реакции на изменения состояния
- Factory для создания сложных компонентов

Пример для Model-View-Controller

```
import tkinter as tk
from tkinter import ttk
class UserModel:
    def __init__(self):
        self.users, self.obs = [], []
    def add_user(self, u, e):
        self.users.append({«username»: u, «email»: e})
        for o in self.obs: o.on_change()
    def add_obs(self, o): self.obs.append(o)
class UserView(ttk.Frame):
    def __init__(self, parent, ctrl):
        super().__init__(parent)
        ttk.Entry(self).pack(pady=2); self.entry1 = ttk.Entry(self); self.entry1.pack(pady=2)
        ttk.Entry(self).pack(pady=2); self.entry2 = ttk.Entry(self); self.entry2.pack(pady=2)
       ttk.Button(self, text=»Add», command=lambda: ctrl.add_user(self.entry1.get(), self.entry2.get())).pack()
        self.listbox = tk.Listbox(self); self.listbox.pack(fill=tk.BOTH, expand=True)
    def update_list(self, users):
        self.listbox.delete(0, tk.END)
        for u in users: self.listbox.insert(tk.END, f»{u[‘username’]} - {u[‘email’]}»)
class UserController:
    def __init__(self): self.model, self.view = UserModel(), None
    def set_view(self, v): self.view = v; self.model.add_obs(self)
    def add_user(self, u, e): u and e and self.model.add_user(u, e)
    def on_change(self): self.view and self.view.update_list(self.model.users)
class Settings:
    def __init__(self):
        self._theme, self._lang, self.obs = «light», «en», []
    def add_obs(self, o): self.obs.append(o)
    def notify(self, **k): [o.update(**k) for o in self.obs]
    @property
    def theme(self): return self._theme
    @theme.setter
    def theme(self, v): self._theme = v; self.notify(theme=v)
    @property
    def language(self): return self._lang
    @language.setter
    def language(self, v): self._lang = v; self.notify(language=v)
def run_mvc():
    root = tk.Tk(); root.title(«MVC»)
    c = UserController(); v = UserView(root, c); c.set_view(v)
    v.pack(fill=tk.BOTH, expand=True); root.mainloop()

def run_observer():
    root = tk.Tk(); root.title(«Observer»); root.geometry(«300x200»)
    s = Settings()
    f = tk.Frame(root, height=150); f.pack(fill=tk.BOTH, expand=True)
    lbl = tk.Label(root, text=»Theme: light, Lang: en», relief=tk.SUNKEN)
    lbl.pack(fill=tk.X, side=tk.BOTTOM)
   s.add_obs(type(‘’,(),{‘update’: lambda **k: f.config(bg=»black» if k.get(‘theme’)==»dark» else «white»)})())
        s.add_obs(type(‘’,(),{‘update’:  lambda  **k:  lbl.config(text=f›Theme:  {k.get(‘theme’,  ‘light’)},  Lang:
{k.get(‘language’, ‘en’)}»)})())
    for t, cmd in [(«Light», ‘light’), («Dark», ‘dark’), («EN», ‘en’), («RU», ‘ru’)]:
        attr = ‘theme’ if t in [‘Light’, ‘Dark’] else ‘language’
        tk.Button(root, text=t, command=lambda a=attr, v=cmd: setattr(s, a, v)).pack(side=tk.LEFT)
    root.mainloop()
def main():
    root = tk.Tk(); root.title(«Patterns»); root.geometry(«250x100»)
    for t, cmd in [(«MVC», run_mvc), («Observer», run_observer), («Exit», root.destroy)]:
        tk.Button(root, text=t, command=cmd, width=15).pack(pady=2)
    root.mainloop()

if __name__ == «__main__»: main()
```

Заключение

На основе проведенного анализа можно констатировать, что Tkinter занимает уникальную и устойчивую нишу в современной экосистеме инструментов для разработки клиентской части технологических продуктов. Несмотря на появление множества альтернативных фреймворков, эта библиотека сохраняет свою практическую значимость благодаря сбалансированному сочетанию простоты, надежности и функциональности. Её ключевые преимущества, такие как нулевой порог вхождения и быстрое освоение, делают Tkinter идеальным решением для образовательных целей и быстрого прототипирования. Минимальные требования к инфраструктуре и встроенная доступность в стандартной поставке Python устраняют сложности с развертыванием и обеспечением совместимости, что критически важно для внутренних инструментов и утилит сопровождения.

Кроссплатформенность библиотеки обеспечивает единообразие поведения приложений в различных операционных средах, что значительно сокращает затраты на тестирование и поддержку. При этом Tkinter демонстрирует достаточную гибкость для реализации сложных архитектурных паттернов, включая MVC, Observer и Factory, что позволяет создавать хорошо структурированные и поддерживаемые приложения. Однако важно отметить, что выбор Tkinter должен быть осознанным и соответствовать конкретным задачам проекта. Для публичных коммерческих продуктов с высокими требованиями к современному пользовательскому опыту более оправданным может быть использование других решений. Но в сценариях, где приоритетом являются скорость разработки, минимальные зависимости и стабильность работы, Tkinter продолжает оставаться оптимальным выбором. Таким образом, библиотека не только сохраняет свою актуальность, но и представляет собой ценный инструмент в арсенале разработчика, особенно в специфических предметных областях, где глубоко интегрирована Python-экосистема.

Список литературы

1. Есилевский С. Знакомство с TCL/TK // Системный администратор. 2021. URL: https://elibrary.ru/item.asp?id=46486849 (дата обращения: 15.12.2025).

2. Нехорошева Е. М., Конецкая Е. А. Графические возможности Ruby // Информационно-телекоммуникационные системы и технологии: материалы Всероссийской научно-практической конференции. Кемерово: Кузбас. гос. техн. ун-т им. Т. Ф. Горбачева, 2021. С. 99–101. URL: https://elibrary.ru/item.asp?id=49602973 (дата обращения: 15.12.2025).

3. Использование виджета Treeview в Tkinter, вложенные элементы // PythonRu. 2021. URL: https://pythonru.com/uroki/vidzhet-treeview-tkinter-22 (дата обращения: 15.12.2025).

4. Python Tkinter Tutorial // GeeksforGeeks. 2017. URL: https://www.geeksforgeeks.org/python/python-gui-tkinter/ (дата обращения: 15.12.2025).

5. Tkinter. Программирование GUI на Python / Светлана Шапошникова (plustilino). 2021.

ГИБРИДНАЯ АРХИТЕКТУРА AI-АССИСТЕНТА ДЛЯ УПРАВЛЕНИЯ ФИНАНСАМИ НА ОСНОВЕ МЕТОДОВ МАШИННОГО ОБУЧЕНИЯ И ПОВЕДЕНЧЕСКОЙ ЭКОНОМИКИ

Денисов Е. Д.

ФГБОУ ВО «Российский экономический университет им. Г. В. Плеханова», Москва, e-mail: jenick47@gmail.com

Научный руководитель: Микрюков А. А.

Введение

Современные приложения для учета финансов (например CoinKeeper, Дзен-мани) предназначены для решения задач визуализации динамики, но при этом не обеспечивается коррекция глубинных моделей поведения. Более 70% решений о покупках являются импульсными или иррациональными, обусловленными когнитивными искажениями, такими как предпочтение сиюминутной выгоды долгосрочной цели, либо покупка «актива» под влиянием новостного фона, а не самостоятельного анализа. Идея предлагаемого подхода заключается в создании цифрового коуча для принятия рационального решения субъектом за счёт синтеза метода машинного обучения (для анализа паттернов) и модели поведенческой экономики для их коррекции. В результате сформирована концептуальная модель алгоритма.

Теоретический анализ рынка

Проведенный анализ существующих решений показывает их принципиальное ограничение, они функционируют как пассивные регистраторы финансовых операций, не оказывая активного воздействия на поведенческие паттерны пользователя. Такие системы предоставляют историческую статистику, но не предотвращают повторение финансовых ошибок.

Возникает противоречие между объективной потребностью пользователей в повышении финансовой дисциплины и неспособностью традиционных приложений преодолеть психологические барьеры, препятствующие рациональному финансовому поведению. Данное противоречие может быть разрешено за счет создания модели, которая фиксирует финансовые операции и активно корректирует поведение пользователя на основе принципов поведенческой экономики.

Концептуальная модель

В ходе исследования была разработана концептуальная модель AI ассистента на основе алгоритма поддержки принятия решений, что реализовано в виде мобильного приложения с серверной частью и состоит из 4 основных модулей (рисунок).