УДК 004.514:003

Семиотическая модель графического пользовательского интерфейса

Цыбульский В. М.
Учреждение образования «Витебский государственный университет имени П. М. Машерова», Витебск

Статья посвящена графическим пользовательским интерфейсам как самому распространенному способу управления электронно-вычислительными устройствами. Предпринята попытка формализации языка интерфейса, которая направлена на разработку более эффективных инструментов для проектирования сложных, нагруженных смыслом, коммуникационных систем.

Представлена семиотическая модель графического пользовательского интерфейса, описана его структура, выделены базовые составляющие объекта коммуникации на основе треугольника Г. Фреге. В рамках системы рассмотрены базовые графические элементы, ментальные модели пользователя и запрограммированные функции. Используя человеко-центрический подход в исследовании интерфейсов систематизированы практические знания, сформулирована четкая и понятная модель интерфейса как знаковой системы, подготовлена теоретически-обоснованная база для дальнейшей разработки и изучения данного вопроса.

Ключевые слова: *графический пользовательский интерфейс (GUI), человеко-компьютерное взаимодействие (HCI), семиотическая модель, человеко-центрированный дизайн (HCD), графический язык, структура интерфейса, знаковые системы.*

(Искусство и культура. – 2018. – № 3 (31). – С. 67–76)

Semiotic Model of the Graphical User Interface

Tsybulski V. M.
Educational Establishment "Vitebsk State P. M. Masherov University", Vitebsk

The article is devoted to the graphical user interface as a most common way of human-computer interaction. Attempt is made to formalize the interface language aimed to develop more effective tools in designing communicative systems with a high semantic load.

The article presents a semiotic model of the graphical user interface; its structure is described; basic components of the object of communication on the basis of G. Frege meaning triangle basic elements are distinguished. Within the system basic graphical elements, user's mental models and programmed functions are considered. With the help of the human-centered design in the interface study, practical groundwork was brought into system, an understandable sign system model of the interface was defined and a theoretically rationale base for further study and research was developed.

Key words: *graphical user interface (GUI), human-computer interaction (HCI), semiotic model, human-centered design (HCD), graphical language, interface structure, sign systems.*

(Art and Cultur. – 2018. – № 3 (31). – P. 67–76)

Ранние подходы к проектированию интерфейсных языков были предприняты еще в 1990-х годах профессором кафедры информатики Папского Католического университета Рио Де Жанейро К. С. Де Суза [1]. Тогда же были разработаны базовые концепции применения семиотики в HCI, получившие название «Семиотическое проектирование» (Semiotic Engineering). Идея исследований заключалась в использовании метаязыка как средства проектирования интерфейса дизайнером и основного языка интерфейса, взаимодействующего с конечным пользователем. Теория охватывала культурологические и социологические аспекты применения семиотики к проектированию. На практике языковый подход к проектированию интерфейса описывает Илья Бирман в курсе лекций Студии А. Горбунова [2], а также в книге «Пользовательский интерфейс» [3]. Тем не менее попыток классификации элементов и выделения четкой структуры интерфейса в рамках семиотической системы до сегодняшнего дня не предпринималось.

Цель статьи – выделить и проанализировать семиотические аспекты в графических пользовательских интерфейсах, а также применяя семиотическую модель к структуре построения интерфейса, рассмотреть его базовые элементы.

Известный семиотик и культуролог XX века Ю. М. Лотман под семиотикой понимал науку о коммуникативных системах и знаках, используемых в процессе общения [4]. На этом основании можно сделать заключение,

Адрес для корреспонденции: e-mail: vladtsm@gmail.com – В. М. Цыбульский

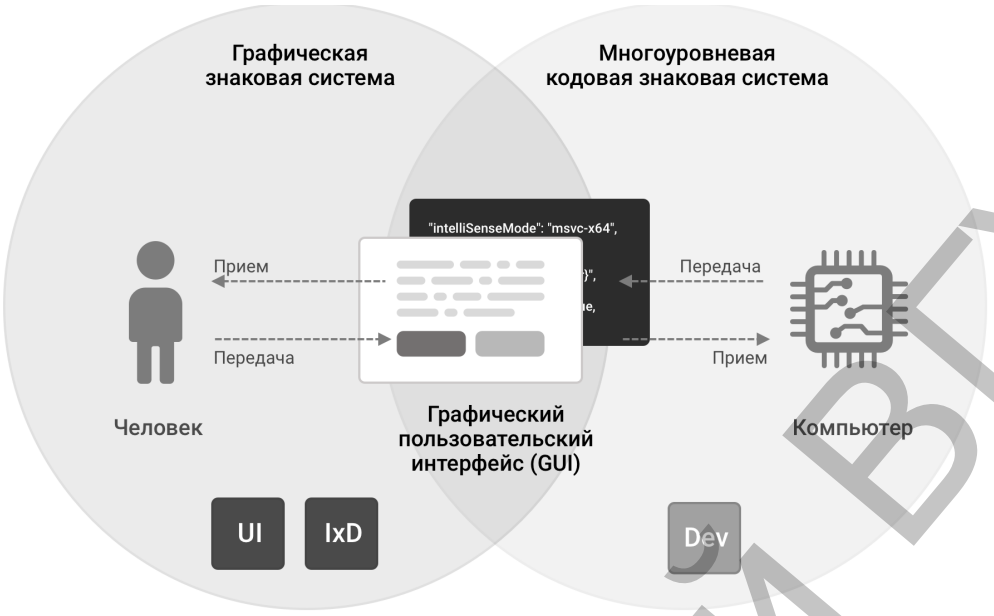

Рис. 1. ***Дуальность коммуникационной системы интерфейса***

что интерфейс, даже в самом широком его понимании (в том числе API и аппаратный интерфейс), также представляет собой коммуникационную систему – язык, транслятор, передающий информацию от одного объекта к другому.

Как в любом языке, в интерфейсах существуют свои правила и закономерности, при нарушении которых передача информации либо затрудняется, либо становится вовсе невозможной. Сообщения должны, с одной стороны, быть максимально емкими (краткими), увеличивая таким образом скорость восприятия и реагирования на информацию, с другой – максимально точно описывать предмет, исключая ошибки интерпретации сообщения. Первое, как правило, «размывает» и делает сообщение неоднозначным, второе увеличивает размер сообщения. Соблюдение баланса между данными категориями является основополагающей задачей как в семиотике, так и в проектировании взаимодействия.

Как известно, интерфейс состоит из двух частей (рис. 1) [5]. Одна часть передает и принимает сигналы от человека, используя графический язык коммуникации, вторая – обеспечивает обмен данными между интерфейсом и компьютером, используя машинные коды. Сам же интерфейс выполняет роль интерпретатора, транслятора с одного языка в другой. В данном исследовании будет использован человеко-центрированный подход к структуре интерфейса условия и требования к которому описаны международным стандартом ISO 9241-210 [6]. Таким образом, принимая человека за основополагающее звено в процессе коммуникации, мы будем рассматривать только первую часть интерфейса, которая является объектом интереса дизайнеров и проектировщиков. В свою очередь, вторая часть – поле деятельности разработчиков программного обеспечения. Тем не менее некоторые аспекты разработки ПО, что обуславливают неоднозначность интерпретации интерфейсом сигналов в обе стороны, также будут рассмотрены в данном исследовании.

В процессе взаимодействия с интерфейсом можно выделить два потока: прием и передача информации. На практике в графических пользовательских интерфейсах для реализации передачи информации интерфейсу (управления им) применяются контролы, а для приема – индикаторы и медийные средства представления контента – своего рода ответ машины на запросы пользователя, обеспечивая, таким образом, интерактивность всей системы.

Воспринимая интерфейс как коммуникационную систему, можно выделить базовую составляющую интерфейса, которая в контексте семиотики представлена знаком. Для определения базовых характеристик элемента (знака) в интерфейсе была выбрана трехчастная модель немецкого логика, математика и философа, представителя школы аналитической философии Готлоба Фреге [7]. В соответствии

Рис. 2. ***Структура знака в интерфейсе по модели Г. Фреге***

с ней структуру элементов интерфейса можно представить тремя категориями:

• базовый графический элемент (означающее, символ);

• ментальная модель (смысл, концепт);

• запрограммированная функция (значение, денотат) (рис. 2).

Базовый графический элемент интерфейса. Контрол (Control) представляет собой примитив графического интерфейса пользователя, имеющий стандартный внешний вид и выполняющий стандартные действия [8]. По семантическим характеристикам базовые элементы можно подразделить на иконические, индексы и символы. Внешние признаки графического элемента определяют «считываемость» знака пользователем. Их важными характеристиками являются цвет, форма и контекст – расположение объекта относительно других знаковых форм. Ошибки распознавания графических элементов или их системы вносят неоднозначность в понимание функции объекта, что приводит к неверно выстроенным ментальным моделям в процессе взаимодействия пользователя с интерфейсом. В процессе развития интерфейса пользователи привыкали к определенным внешним характеристикам элементов со сходной семантикой на разных платформах, по которым сегодня можно выделить вполне определенные группы базовых графических элементов в GUI.

Первой и самой распространенной группой базовых графических элементов в знаковой системе являются триггеры. Типичный представитель триггера в физическом мире – курок (прообраз современной кнопки). Триггер способен воспроизводить однозначное действие «запуск». Триггеры представлены: кнопками (Button), иконками (Icon Button), гипертекстовыми ссылками (Hyperlink) (рис. 3). Разнообразие функций триггера расширяется за счет использования различных кнопок мыши: одиночных, двойных и тройных щелчков; на устройствах с сенсорными экранами тапом (Tap) или сильным нажатием (3D Touch) и пр.

Рис. 3. ***Типовые варианты триггера: кнопка, ссылка, кнопки с иконкой***

Во вторую группу включены селекторы и меню. Селектор и меню являются метафорой переключателя в физическом мире. Элементы данной группы предоставляют пользователю выбор из двух или большего количества категорий в разных конфигурациях и могут быть репрезентированы большим многообразием компонентов (рис. 4): Toggle Switch (используется как переключатель двух состояний), Radio Button (выбор только одного варианта

из набора), Checkbox (выбор/активация нескольких вариантов из набора категорий), Tabbs (выбор вкладки, переключение контекстов), Dropdown Lists (выбор только одного варианта из набора).

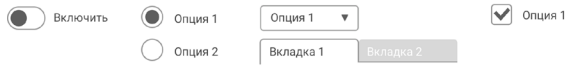

Рис. 4. ***Варианты селекторов: Toggle Switch, Radio Buttons, Dropdown List, Tabbs, Checkbox***

Дискретный селектор – это своеобразная метафора регулятора в физическом мире, которая предоставляет пользователю выбор дискретного многообразия вариантов из множества. Представители дискретных селекторов следующие: Slider (линейное представление), Scrollbar (линейное представление), Rotary slider (концентрическое представление), Image Carousel (линейное представление изображений) (рис. 5).

Рис. 5. ***Варианты дискретных селекторов: Slider, Rotary Slider, Scroll Bar***

Поля ввода формируют третью группу. Они предназначены для получения информации от человека машиной, используя естественный, математический или символьный язык. Поля ввода –графический Text Field элемент, который в зависимости от контекста и программных ограничений может принимать любой из вышеперечисленных языков. Таким образом, поле может служить для поиска, ввода паролей, цифровых или текстовых данных (рис. 6).

Рис. 6. ***Варианты полей ввода: поле ввода пароля, поле ввода числового значения, поле поиска***

Индикаторы составляют четвертую группу и являются для человека средством обратной связи. Они позволяют получать информацию о статусе происходящего процесса или состояния объекта. Среди многообразия индикаторов выделим следующие: Notifications (уведомления), Progress bars (индикаторы состояния), Progress spinners (круговые индикаторы загрузки), Tool Tips (всплывающие подсказки), Counters (счетчики) и др. (рис. 7).

Рис. 7. ***Варианты индикаторов: индикатор состояния GSM сети, индикатор заряда батареи, числовой индикатор, индикатор сообщений, языка и уровня громкости***

Стоит однако оговорить, что конечного набора элементов не существует, могут появляться новые. Тем не менее каждый элемент попадает под одну из категорий и выполняет одну из четырех групп функций. Существуют комбинированные знаковые формы, которые в данной работе мы будем рассматривать как виджеты. Яркими представителями таких систем являются поля ввода с проверкой содержания (Validation) и кнопкой подтверждения, а также редактируемые выпадающие меню, совмещающие в себе поле ввода и селектор (рис. 8).

Рис. 8. ***Комбинированные знаковые формы: поле ввода с кнопкой подтверждения и проверкой содержания, редактируемый Dropdown List***

Термин «виджет» (Widget) был введен в американский язык в 1920 году как понятие, описывающее любое полезное электронное устройство. В компьютерную среду он попал благодаря проекту интерфейса MIT и IBM «Project Athena» в 1988 как сокращение от «window gadget». В данной модели виджеты представляют собой знаковую систему из набора базовых элементов. Будучи знаками (со своим собственным концептом и денотатом), они не выражают сумму семантических смыслов своих элементов. Возможно провести параллель с естественным языком человека, где предложение не является совокупностью смыслов составляющих его слов. Виджеты – синтаксические конструкции,

которыми занимается соответствующий раздел семиотики – синтактика. Так как синтаксис интерфейса является отдельной обширной темой для изучения, поэтому в качестве примера виджета мы возьмем только форму регистрации (рис. 9) [12].

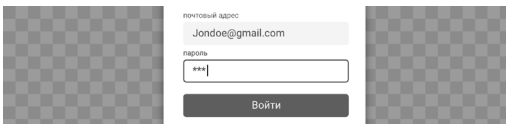

Рис. 9. ***Форма логина. Запрос на введение почтового адреса, пароля и кнопка подтверждения (триггер)***

Для построения грамматически правильного языка дизайнеры опираются не только на законы формальной композиции, но и на особенности восприятия человеком визуальной структуры любой графической системы. Такие принципы были достаточно подробно сформулированы и описаны в гештальтпсихологии еще в 1912 году К. Коффкой, М. Вертгеймером и В. Кёлером [10]. К основным принципам расположения элементов в виджетах, используемым в проектировании интерфейсов, следует отнести: близость, сходство, общую зону, непрерывность, а также закон фигуры и фона.

Принцип близости описывает особенность человека воспринимать близко расположенные элементы, как элементы объединенные одним контекстом. В интерфейсах данный принцип используется в комбинированных графических элементах для разбиения формы на категории, а также для общей организации структуры элементов в интерфейсе (рис. 10). Для применения этого принципа выбираются последовательности, которые определены конкретным общепринятым синтаксисом или контекстом.

Рис. 10. ***Схема принципа близости и пример двух полей, объединенных этим принципом***

При помощи принципа сходства в интерфейсах реализуются группы меню, закладки, списки (рис. 11). В процессе взаимодействия с интерфейсом данный принцип реализуется в элементах выполняющих одну и ту же функцию в разных частях интерфейса. Например, кнопка «закрыть» во всех окнах находится всегда в верхнем правом/левом углу, а полоса прокрутки контента располагается справа или внизу и всегда прокручивает текущую страницу.

Рис. 11. ***Схема принципа сходства. Пример расположения кнопки «закрыть», имеющей стандартный вид и расположенной всегда в одном и том же месте окна***

Общая зона объединяет элементы интерфейса, заключая их в замкнутой области. Ярким примером использования данного принципа являются всплывающие окна (Popup), которые представляют пользователю отдельные окна с содержащимися в них базовыми элементами, объединенными одним контекстом (рис. 12). На персональных компьютерах любая прикладная программа демонстрирует этот принцип, так как ее содержание заключено в рамку «окна».

Рис. 12. ***Схема принципа общей зоны и всплывающего окна, построенного на этом принципе***

Принцип непрерывности, в большей степени, касается контента подаваемого в динамике. Прокручивание страниц в интерфейсе позволяет пользователю воспринимать содержимое не как набор не связанных между собой картинок, а как объединенную контекстом общую систему (рис. 13). Данный принцип также выражается в разного рода жестах (gestures) на мобильных устройствах: листании, смахивании (Swipe), увеличении содержания (Pitch to zoom)

Рис. 13. ***Схема принципа непрерывности и пример применения его при прокручивании контента***

Закон фигуры и фона является фундаментальным правилом разделения элементов на важные и не важные в данный момент. Для реализации принципа применяются методы изменения формы, цвета и контраста элемента или набора элементов, размывания или затемнения содержания позади виджета (рис. 14).

Рис. 14. ***Схема закона фигуры и фона и пример***

Таким образом, символ знака выполняет идентификационную роль и может быть выражен большим многообразием управляющих элементов, которые представляют собой алфавит интерфейса. Способность использовать этот алфавит для коммуникации зависит от правильного распознавания пользователем базовых графических элементов системы и их синтаксических структур. Для этих целей проектировщик обязан использовать четкие критерии выбора внешних характеристик элементов и принципы человеческого восприятия.

Ментальная модель пользователя. В проектировании взаимодействия она является одной из определяющих категорий элемента (знака). На базе ментальной модели выстраивается вся коммуникация пользователя с системой: логика дальнейших действий, предсказуемость системы и общее понимание концептуальной модели программы. По словам американского ученого в области когнитивистики, дизайна и пользовательской инженерии Дональда Нормана: «Ментальные модели – это то, что сидит у людей в головах, тот процесс, который направляет их при использовании объектов…» [11].

Ментальная модель – это интуитивное понимание принципов работы объекта или системы, основанное на прошлом опыте человека, имеющейся информации и здравом смысле. Ментальные модели – знания и представления о реальности в виде систем взаимосвязанных фактов и причинно-следственных связей. В психологии этому компоненту аналитической деятельности соответствуют ментальные репрезентации, понимаемые как актуальный умственный образ того или иного конкретного события или объекта.

В 80-х годах прошлого столетия принципы ментальных моделей и когнитивной семантики получили свое развитие в книге Филипа Джонсон-Лэрда «Ментальные модели» [12]. Книга содержит анализ вопросов, связанных со значением языковых выражений и пониманием языка. По его словам, изучение мышления не может быть ограничено рамками какой бы то ни было одной дисциплины – требуются объединенные усилия экспериментальной психологии, лингвистики и теории искусственного интеллекта, синтез их идей и методов. Понимание, как считает Джонсон-Лэрд, «определяется знанием и уверенностью: человек понимает (до некоторой степени) то или иное явление тогда, когда он имеет представление о его причинах и следствиях, внутренней структуре и связях с другими явлениями, о способах его предсказания и предотвращения, о путях его инициации, управления им и контроля за ним» [13]. Таким образом, психологическое ядро понимания составляет наличие у человека некой «рабочей модели» соответствующей сущности или явления.

Ментальные модели непостоянны. Наблюдая за объектом в течение определенного периода времени, мы можем заметить его изменения. И каждый раз, когда человек пытается сделать вывод относительно меняющегося объекта, различные результаты наблюдения могут разрушать существующую ментальную модель и создавать новую. Как отмечает английский философ и психолог Кеннет Крейк, ментальные модели по своей природе являются неполными и постоянно развивающимися.

Таким образом, достаточно сложно определить точную ментальную модель пользователя в процессе взаимодействия с интерфейсом. Если понимание человеком поведения элементов в системе соответствует тому, какие функции эти элементы выполняют, можно утверждать, что интерфейс является интуитивно понятным. Что касается взаимодействия с программным обеспечением, пользователи ожидают увидеть определенную последовательность действий в соответствии с их прошлым опытом и ожиданиями. В качестве примера такой последовательности можно вернуться к форме регистрации рассмотренной выше (рис. 9). Пользователь вводит необходимые данные, нажимает кнопку подтверждения, получает ссылку для активации на почту, затем переходит по ней, чтобы активировать свой аккаунт. Большинство пользователей ожидают увидеть в процессе регистрации именно такие шаги и держат в голове ментальную модель того, какой должна быть процедура регистрации.

Формирование ментальной модели в интерфейсах происходит по следующим принципам:

- наблюдение;
- погружение;
- обучение.

В процессе *наблюдения* сознанием выделяются характеристики окружающих объектов, строится семантика объекта. Таким образом, каждый человек создает свои собственные неповторимые и уникальные ментальные модели.

Погружение – это такой тип пользовательского опыта, при котором пользователь полностью поглощен взаимодействием с системой. Погружение стимулирует его чувства. Так же процесс погружения сравнивают с состоянием «потока» [14]. Человек настолько глубоко концентрируется на задаче, что перестает замечать все вокруг, теряясь во времени. Опыт погружения сам по себе не формирует у человека ментальную модель, он скорее способствует выходу ощущений пользователя на другой уровень. Тем не менее, когда люди сравнивают и противопоставляют новый опыт старому, они начинают представлять себе, каким же опыт должен быть. Это представление и создает ментальную модель. После того, как пользователи сформировали ментальную модель, они будут отвергать опыт, который ей не соответствует.

Процесс приспособления ментальной модели пользователя к концептуальной модели продукта происходит по принципу *обучения*. Следует особо отметить разницу между ментальными и концептуальными моделями. Концептуальная модель – это действующая модель, которую человек получает, знакомясь с дизайном и интерфейсом конкретного продукта. Ментальные же модели люди используют для прогнозирования поведения системы, предмета или интерфейса, и для того, чтобы понять, какие операции с их помощью можно производить.

Культура играет важную роль в формировании человеческого восприятия. Пользователи из другой страны часто сталкиваются с определенными трудностями при взаимодействии с интерфейсами: языковой барьер, социальные нормы, раскладка клавиатуры, проблемы с распознаванием символов и знаков, формат валюты и дат, единицы измерения и даже юридические требования.

Кросс-культурное проектирование призвано обеспечить простоту использования и приятный опыт взаимодействия с продуктом пользователю несмотря на межкультурные барьеры. При проектировании пользовательского взаимодействия в кросс-культурной среде очень важно уделять особое внимание таким ключевым аспектам, как организация контента, навигационная структура, цветовая гамма, типографика, а также и элементам дизайна, как изображения, логотипы, фотографии и анимация.

В процессе проектирования дизайнер учитывает логически продуманную концептуальную модель, чтобы сделать интерфейс понятным потребителю. В действительности, интерфейс никогда не может соответствовать ментальной модели каждого пользователя. Число возможных моделей может достигать многих тысяч. Тем не менее дизайнеры используют обобщенный ментальные модели большинства пользователей, изучая их потребности, склонности, желания и прошлый опыт.

Сегодня существует ряд популярных человеко-центрированных (Human centered design, HCD) подходов к проектированию пользовательского взаимодействия. Они предлагают методы проектирования взаимодействия через ментальные модели пользователей. Алан Купер, известный исследователь проектирования взаимодействия, в конце 90-х годов предложил концепцию «персон» [15]. На основе интервьюирования группы людей он создавал «персону» – собирательный образ этой аудитории. Такой подход помог ему объединить людей с разным менталитетом и воспринимать их как потенциальных потребителей проектируемого продукта. Таким образом, «персонажи» помогают понять контекст, поведение, отношения, потребности, проблемы, тревоги, сомнения, цели и мотивацию пользователей.

Другим подходом в проектировании взаимодействия является активное использование эмпатии. Известный специалист в области юзабилити Уитни Кьюсенбери определяет эмпатию как способность понимать и чувствовать контекст, эмоции, цели и мотивацию другого человека [16]. Длительное погружение в роль целевого пользователя дает возможность оценить контекстные сигналы из окружающей среды, которые позволяют понять связь между вещами для потребителя. Так же дизайнерами используется контекстный подход. Когда нужно изучить пользователей в их естественной среде, контекстные исследования являются эффективным методом для получения необходимой информации. Это метод наблюдения за людьми в конкретной ситуации, который помогает понять и проанализировать принципы принятия решения пользователями.

Следует отметить, что существующие различия в ментальных моделях человека могут привести к более низкой производительности коммуникационной системы, а также последующему разочарованию пользователя процессом взаимодействия с приложением. Чтобы избежать этих негативных последствий, проектировщики интерфейсов должны принимать во внимание ментальные модели потребителей, для которых создается программный продукт.

Запрограммированная функция. Она представляет собой денотат знака, его значение. Ее характеристики определяются исходным кодом программы, и ее реализацией занимаются специалисты по разработке программного обеспечения. В идеальных условиях запрограммированная функция элемента должна максимально соответствовать ментальной модели, сохраняя константность «поведения» в рамках всей семиотической системы.

Структура пользовательского интерфейса со стороны машины представляет собой многоуровневую систему переходных процессов. Можно определить несколько стадий преобразования, которые проходит сигнал коммуникации, прежде чем он преобразуется в понятные вычислительной машине бинарные коды.

Первый, самый высокоуровневый этап преобразования представлен фреймворками (framework) и высокоуровневыми языками программирования (которые впоследствии интерпретируются в более низкоуровневые промежуточные языки и далее компилируются в бинарные коды). В случае с веб-приложениями это могут быть популярные в современном «сайтостроении» высокоуровневые интерпретируемые языки: JavaScript (фреймворки Angular.js, React.js, Backbone.js, Vue.js, Embre.js, Meteor.js), Python (фреймворки Django, Pyramid, TurboGears, Flask, Bottle, Tornado), Rubi (фреймворки Rubi on Rails, Sinatra, Padrino, Hanami, NYNY, Scorched, Cuba) и т. д. Языки для разработки десктоп- и мобильных приложений представлены: Swift, ObjectiveC, C++, C#, Java.

Второй этап преобразования происходит на уровне компилируемых языков. Как правило, сегодня таким языком программирования является Си. На третьем этапе коммуникационный сигнал проходит уровень операционной системы устройства (ОС рассматривается как набор стандартизированных библиотек и драйверов призванных регулировать процесс работы прикладного ПО и перенаправлять их сигналы к физическим устройствам, используя различные API). Последним этапом в процессе преобразований является исполнение бинарных машинных кодов компьютером на аппаратном уровне [17].

Сегодня способ выстраивания архитектуры приложения и написание ее исходного кода в большей степени определены стандартными библиотеками, которые предоставляет операционная система для разработчика прикладного ПО. Таким образом, стандарт определенный системой налагает некоторые ограничения в проектировании логики и интерфейса приложения.

Важно отметить, что в среде разработки ПО наблюдается тенденция движения к унификации средств разработки под разные операционные системы. Примером могут служить кроссплатформенные фреймворки, которые представляют из себя дополнительный уровень абстракции над целевыми платформами. Такой подход обладает рядом очень важных преимуществ для разработчиков и проектировщиков взаимодействия. Увеличение скорости разработки и тестирования продукта или сервиса на разных платформах способствует более быстрому выходу программного продукта на рынок и получению обратной связи от пользователя. Применение единых графических элементов на разных платформах обеспечивает константность восприятия пользователем интерфейса (особенно при разработке серии приложений для разных устройств воспроизводящих одни и те же функции). Не менее важен тот факт, что разработчики используют одинаковый инструментарий в процессе написания кода, что гарантирует в какой-то степени стабильность работы программы и соответствие запрограммированных функций ожидаемым результатам на разных платформах.

Однако, учитывая, что все кроссплатформенные решения – это дополнительный уровень абстракции, введение новых возможностей или технологий разработчиком платформы требует адаптации кроссплатформенного фреймворка к новым технологиям. В современных условиях данный процесс происходит очень медленно. Поэтому кроссплатформенное ПО находится всегда «на шаг позади» от программ, разработанных специально для операционной системы ее «родными» средствами.

На данный момент существует большой выбор фреймворков общего назначения для проектирования графических пользовательских интерфейсов. Большинство из них способны работать на популярных десктоп- и мобильных платформах: MacOS, Linux,

Windows, iOS, Android. Например, Qt является основой интерфейса популярной рабочей среды KDE, входящей в состав многих дистрибутивов Linux. Так же с использованием Qt созданы интерфейс бортового компьютера Tesla Model S, Skype, Telegram и другие. Данный фреймворк обладает всеми основными классами, которые могут потребоваться при разработке интерфейсов: готовыми виджетами, модулями работы с графикой и анимацией, непосредственным доступом к OpenGL/DirectX, интегрированным движком интернет-браузера Chromium, доступом к мультимедиа устройствам и многим другим. Основным же средством проектирования интерфейсов в Qt считается их собственный язык разметки QML, активно развивающийся в последнее время [18].

Другим популярным фреймворком является wxWidgets. В отличие от Qt он позволяет разрабатывать приложения, которые адаптируют внешний вид под стилистику операционной системы (native look), то есть фреймворк использует стандартное API операционной системы для построения графического пользовательского интерфейса. Что, с одной стороны, позволяет создавать приложения с привычным пользователям платформы внешним видом и особенностями взаимодействия, а с другой – приводит к дополнительным трудностям, в случае необходимости реализации одинакового поведения и внешнего вида интерфейса.

Сегодня пользовательские данные хранятся в облаке, синхронизация настроек приложений производится через интернет, через всемирную паутину осуществляется доступ к глобальным информационным данным – сервисы и программное обеспечение уже не могут существовать без выхода в сеть. Становятся популярны веб-приложения. Появляется большое количество готовых решений для разработки приложений, исполняемых в браузере.

Приложения, написанные на связке HTML, CSS, JavaScript, изначально разрабатываемые для использования в браузере, благодаря набору современных библиотек и технологии WebKit способны работать на любых архитектурах и платформах. Таким образом, применение высокоуровневых языков и наличие огромного количества готовых библиотек для реализации интерфейса значительно ускоряют и упрощают процесс разработки ПО. В качестве примера такой технологии можно привести Electron и NW.js.

Оба фреймворка построены на библиотеке Node.js и Chromium браузере, оба поддерживаются крупными компаниями (GitHub и Intel). Electron и NW.js появились в начале этого десятилетия (2013, 2011 – соответсвенно) и предполагают разработку преимущественно на JavaScript. В итоге это приводит к потенциально еще более худшей производительности и потреблению памяти, чем Java. Несмотря на относительную молодость, на обоих инструментариях уже сделано немало приложений: Atom, Visual Studio Code, Slack, WordPress.com desktop app. Наиболее заметным преимуществом таких фреймворков является наибольшая гибкость и отсутствие ограничений в пользовательском интерфейсе – весь интерфейс описывается HTML и CSS. Также Electron не предоставляет защиту исходного кода, а NW.js дает такую возможность, при этом усложняя понимание исходного кода (с последующей потерей производительности).

Существуют значительные ограничения в работе приложений, использующих веб-технологии. Из-за достаточно длинного процесса преобразований на пути к бинарным кодам, понятным компьютеру, высокоуровневые языки веб-программирования не способны напрямую управлять «железом» (Hardware). Диапазон функций ограничивается исключительно уже существующими библиотеками. Таким образом, очевидно преимущество низкоуровневых языков в разработке приложений, использующих новые аппаратные средства или новые технологические решения в интерфейсах.

Кроме чисто технических аспектов, влияющих на процесс проектирования коммуникационной системы человек–компьютер, существуют еще аспекты организационные. При проектировании интерфейса дизайнер взаимодействия должен четко описывать запрограммированную функцию в документации для разработчика. В его задачи не входит просчет всех возможных вариантов поведения элемента или виджета в контексте коммуникационной системы, поэтому проектировщик обязан учитывать не только особенности восприятия и интерпретации объекта пользователем, но еще и уметь описать возможные варианты реагирования элемента в разных контекстных ситуациях. При описании семантики и синтаксиса интерфейса для разработчиков ПО проектировщик также должен учитывать технические ограничения, описанные выше.

Заключение. Коммуникационная система интерфейса представляет собой многоуровневую структуру. В ней присутствуют как графический, так и программный языки. Способы передачи и приема информации

осуществляется по определенным законам и представлены конкретными категориями. Применение трехчастной модели Г. Фреге к структурным элементам интерфейса дало возможность разделить их на три составляющие: базовый графический элемент, ментальную модель и запрограммированную функцию. В свою очередь, структурно-семиотический метод моделирования конструкции GUI позволил выявить группы знаковых форм среди базовых элементов, основываясь на их семантике. Также была определена взаимосвязь между этими группами элементов ментальными моделями человека и запрограммированными функциями.

Следует отметить, что применение семиотических принципов к структуре интереса позволяет систематизировать практические знания, используемые при проектировании графического пользовательского интерфейса, выстроить четкую и понятную модель интерфейса как знаковой системы. Представленный анализ типологии GUI, а также используемых в них видах визуальных коммуникаций, может получить дальнейшее развитие в исследованиях и практике человеко-компьютерного взаимодействия. Последующие разработки в данной области благодаря циклу «задача-артефакт» помогут устранить все неточности модели на практике и дадут возможность эффективно использовать ее при проектировании графических пользовательских интерфейсов.

ЛИТЕРАТУРА

1. Souza, C. S. de de. The Semiotic Engineering of Human-Computer Interaction / C. S. de de Souza. – Cambridge, Mass: The MIT Press, 2005. – 312 c.

2. Пользовательский интерфейс и представление информации [Электронный ресурс] / Дизайн-бюро Артема Горбунова. – Режим доступа: http://bureau.ru/educenter/ui/. – Дата доступа: 30.05.2018.

3. Пользовательский интерфейс. Электронный учебник [Электронный ресурс]. – Режим доступа: http://artgorbunov.ru//projects/book-ui/. – Дата доступа: 30.05.2018.

4. Лотман, Ю. М. Внутри мыслящих миров / Ю. М. Лотман. – СПб.: Издательская Группа «Азбука-Аттикус», 2015. – 415 с.

5. Berkshire encyclopedia of human-computer interaction / ed. W.S. Bainbridge. – Great Barrington, Mass : Berkshire Pub. Group, 2004. – 2 p.

6. ISO 9241-210:2010 - Ergonomics of human-system interaction – Part 210: Human-centred design for interactive systems [Electronic resource]. – Mode of access: https://www.iso.org/standard/52075.html. – Date of access: 07.06.2018.

7. Фреге, Г. Логика и логическая семантика / Г. Фреге; пер. Б. Бирюков. – Изд. монография. – Москва: URSS, 2012. – 595 с.

8. Affairs, A. S. for P. User Interface Elements [Electronic resource]. – Mode of access: /how-to-and-tools/methods/user-interface-elements.html. – Date of access: 05.03.2018.

9. Интерфейсы без шелухи: помогаем интерфейсам стать человечнее [Электронный ресурс] : Интерфейсы без шелухи / dangry.ru. – Режим доступа: https://dangry.ru/sona/interface/syntax/. – Дата доступа: 30.05.2018.

10. Koffka, K. Principles of Gestalt Psychology / K. Koffka. – Psychology Press, 1999. – 736 p.

11. Norman, D. The Design of Everyday Things: Revised and Expanded Edition / D. Norman. – Hachette UK, 2013. – 346 p.

12. Johnson-Laird, P. N. Mental models: towards a cognitive science of language, inference, and consciousness : Cognitive science series / P. N. Johnson-Laird. – Изд. 6. print. – Cambridge: Harvard Univ. Press, 1995. – Вып. 6. – 513 с.

13. Johnson-Laird, P. N. How We Reason / P. N. Johnson-Laird. – Oxford: Oxford University Press, 2008.

14. Csikszentmihalyi, M. Flow: The psychology of optimal experience: Harper Perennial Modern Classics / M. Csikszentmihalyi. – Pub. Nachdr. – New York: Harper [and] Row, 2009. – 303 с.

15. Cooper, A. About face: the essentials of interaction design, 4th edition / A. Cooper. – Pub. 4th edition. – Indianapolis, IN: John Wiley and Sons, 2014.

16. Quesenbery, W. Global UX: design and research in a connected world / W. Quesenbery, D. Szuc. – Waltham, MA: Morgan Kaufmann, 2012. – 238 p.

17. McConnell, S. Code Complete: A Practical Handbook of Software Construction, Second Edition / S. McConnell. – Pub. 2 nd edition. – Redmond, Wash : Microsoft Press, 2004. – 960 p.

18. Summerfield, M. Advanced Qt Programming: Creating Great Software with C++ and Qt 4 / M. Summerfield. – Pub. 1st. – Prentice Hall., 2010. – 560 p.

Поступила в редакцию 26.06.2018 г.