Составные части программного интерфейса.
Элементы управления.

В этой части описаны все кирпичи, из которых состоит любая программа или сайт. Это окна, элементы управления и стандартные сценарии поведения системы.

Кнопки

Кнопкой называется элемент управления, всё взаимодействие пользователя с которым ограничивается одним действием – нажатием. Эта формулировка, кажущаяся бесполезной и примитивной, на самом деле очень важна, поскольку переводит в гордое звание кнопок многие элементы управления, которые как кнопки по большей части не воспринимаются.

Командные кнопки

Нажатие на такую кнопку запускает какое-либо явное действие, поэтому правильнее называть такие кнопки «кнопками прямого действия». С точки зрения разработчика ПО, для настольных систем, командные кнопки являются чрезвычайно простыми. На современных самолетах, в основном на потолочных панелях, используются кнопки со встроенным точечным дисплеем, отражающим текущее состояние по ее функциональному назначению. Изменение назначения кнопки контролируется бортовым компьютером в процессе всего полета.

С точки зрения когнетики, чем понятнее надписи, тем лучше их видно. Проблема только в том, что эта надпись закрывается во время нажатия кнопки, когда она как раз и нужна. Существует правило: - «Чем меньше кнопок, тем лучше». Правда, не всегда, так как кнопки прямого управления с точки зрения пользователя лучше, чем кнопки, раскрывающие меню, поэтому меньшее число кнопок и более простой вид лицевой панели не всегда являются лучшим решением. Это справедливо не только для графических интерфейсов, но и для панелей и щитков управления различных приборов, пультов управления, приборных досок и т. п.

Иная ситуация в Интернете, где отсутствие операционной системы (откуда приходят элементы управления) и простота создания новых типов кнопок (это чуть ли не единственный элемент, с которым вообще удается что-либо сделать) привели к тому, что нестандартные кнопки не создает только ленивый.

В то же время, этот самый простой элемент управления имеет больше всего тонкостей. Но по порядку.

В Интернете кнопка должна быть оформлена как текстовая ссылка, если она перемещает пользователя на другой фрагмент текста, и как кнопка – если она запускает действие

Размеры и поля. Как вы уже знаете, чем больше кнопка, тем легче попасть в нее курсором. Это правило по мере сил всеми соблюдается, во всяком случае, кнопок размером 5 на 5 пикселей уже практически не встретишь. Однако помимо простоты нажатия на кнопку есть другая составляющая проблемы: пользователю должно быть трудно нажать не на ту кнопку.

Добиться этого можно либо изменением состояния кнопки при наведении на неё курсором, либо установлением пустого промежутка между кнопками.

Первый способ приобрел существенную популярность в Интернете, второй – в обычном ПО. Он, кстати, более эффективен: одно дело, когда пользователь промахивается мимо кнопки и совсем другое – если, промахнувшись, он ещё и ошибочно нажимает на другую кнопку. Ни тот, ни другой способы не обеспечивают стопроцентной надежности, так что при прочих равных использовать стоит оба.

Считать экранную кнопку нажатой нужно не тогда, когда пользователь нажимает кнопку мыши, а курсор находится на кнопке, а тогда, когда пользователь отпускает нажатую кнопку мыши, курсор в это время находится на экранной кнопке, и находился на ней еще, когда кнопка мыши нажималась

Другой составляющей проблемы размера кнопок в Интернете является несоответствие видимой площади кнопки её действующей площади. В последнее время, кнопки часто реализуют посредством окрашенных ячеек таблицы, в которых размещается текст, являющийся гипертекстовой ссылкой. Проблема заключается в том, что пользователи воспринимают кнопкой всю ячейку, хотя реально «нажимается» лишь малая её часть.

Объем. Кнопка должна (или не должна) быть пользователем нажата. Соответственно, пользователю нужно как-то сигнализировать, что кнопка нажимаема. Лучшим способом такой индикации является придание кнопке псевдообъема, т.е. визуальной высоты. С другой стороны, этот объем плох тем, что при его использовании возникает рассогласование между обликами кнопок прямого и непрямого действия. Разумеется, никто не отменял ещё и тот факт, что псевдообъем кнопок, вообще говоря, в существенной степени есть визуальный шум. Еще с одной стороны, зачастую возникает необходимость максимально повышать шансы нажатия пользователем какой-либо отдельной кнопки (например, «О компании»), в этих случаях псевдообъем этой кнопки (при прочих плоских) сильно повышает вероятность нажатия.

Направление теней во всех элементах управления должно быть одинаковым: снизу справа

Состояния. Кнопка должна как-то показывать пользователям свои возможные и текущие состояния. Количество состояний довольно велико, при этом наборы возможных состояний в ПО и в Интернете значительно различаются. Например, кнопка в Windows может иметь пять состояний: нейтральное, нажатое, нейтральное с установленным фокусом ввода, состояние кнопки по умолчанию и заблокированное состояние. В Интернете обычно используют меньший набор состояний: нейтральное, готовое к нажатию (onMouseOver) и активное (в случаях, когда набор кнопок используется для индикации навигации). Нажатое и заблокированное состояние используются очень редко, а «нейтральное с установленным фокусом ввода» старается, как может, создать браузер.

Вообще говоря, обычно, чем больше набор состояний, тем лучше. Но главное не это, а отсутствие дублирования состояний: не должно быть разных состояний, выглядящих одинаково. Также очень важно делать заблокированные состояния

действительно заблокированными: так, например, в Интернете очень часто встречаются кнопки, нажатие на которые открывают ту же самую страницу, т.е. нажатие которых возможно, но бесполезно. Такие кнопки должны не только выглядеть заблокированными (менее яркими и значительными, нежели обычные), но и не нести гипертекстовых ссылок.

Никогда не удаляйте элементы, которые нельзя нажать, взамен этого делайте их заблокированными

Текст и пиктограммы. Все руководства по разработке интерфейса с изумительным упорством требуют снабжать командные кнопки названиями, выраженными в виде глаголов в форме инфинитива. Разработчики же интерфейса с не менее изумительным упорством не следуют этому правилу. Аргументов у них два: во-первых, все так делают, значит, это есть стандарт и ему нужно следовать, во-вторых, нет времени придумывать название.

Оба аргументы сильны. Действительно, стандарт. Действительно, нет времени. Но есть два контраргумента: во-первых, это не столько стандарт, сколько стандартная ошибка, во-вторых, думать можно и по дороге домой.

Если второй контраргумент особых объяснений не требует, то сущность первого полезно объяснить. Кнопка, запускающая действие, недаром называется командной. С её помощью пользователи отдают системе команды. Команда же в русском языке формируется посредством глагола в повелительном наклонении

Помимо этого, у глагольных кнопок есть одно большое достоинство. По ним понятно, какое действие произойдет после нажатия. Это позволяет как-то разграничить диалоговые окна в сознании (поскольку разные диалоговые окна получают разные кнопки). В результате, из-за увеличения степени уникальности фрагментов системы, обучаться системе получается лучше, нежели с кнопками, одинаковыми везде. Более того, вкупе со строкой заголовка окна, глагольные кнопки создают контекст, что очень полезно при возвращении к прерванной работе. Оказывается возможным не рассматривать *всё* диалоговое окно, чтобы узнать, на каком действии задача была прервана – достаточно просто прочесть надпись на кнопке.

Таким образом, следует всемерно избегать создания кнопок с ничего не говорящим текстом, поскольку такой текст не сообщает пользователям, что именно произойдет после нажатия кнопки. При этом есть одна тонкость. Существующие интерфейсы заполнены терминационными кнопками: «Ввод» (Ок), «Отмена» (Cancel) и «Применить» (Apply), что, собственно говоря, и позволяет разработчикам ссылаться на стандарт. Эти кнопки плохи.

С первой кнопкой понятно – это не глагол, а значит, кнопка плоха. Кнопка одна и та же во всех диалогах, значит всё ещё хуже. Вторая, хоть и почти глагол, плоха, поскольку не дает контекста (к тому же отглагольные существительные воспринимаются медленнее, чем соответствующие глаголы). Главный её недостаток, впрочем, заключается в том, что её, как правило, нечем заменить, так что приходится пользоваться ею. Третья, будучи и глагольной, и сравнительно уникальной, имеет другой недостаток: она почти всегда используется неправильно. На ней написано «Применить», но на самом деле её значение совсем иное. Разберем это подробнее.

Как правило, разработчики создают диалоговое окно, внизу которого располагают три кнопки: Ок, «Применить» и «Отмена» (прямо-таки триединство ошибки). Проблемы наступают тогда, когда пользователь делает что-либо в диалоговом окне и начинает думать, какую кнопку ему нужно нажать. Предположим, он всем доволен и нажимает кнопку ОК. Не считая слабо переданного контекста, все довольно хорошо. Все довольно неплохо, если пользователь нажмет кнопку Отмена – его команды просто не будут обработаны системой. А теперь предположим, что пользователь нажал кнопку «Применить». Система выполняет команду пользователя и меняет данные. Начинается самое интересное: теперь кнопка ОК не делает ничего (команда-то уже обработана), помимо закрытия окна. Т.е. эту кнопку в данном состоянии нужно переименовывать в

Закрыть. Более того. Кнопка Отмена после нажатия кнопки Применить тоже начинает врать пользователю: она не отменяет действие, но просто закрывает окно. Таким образом, если делать интерфейс полностью однозначным, получается гадость: последовательность кнопок Ок, Применить и Отмена после нажатия кнопки Применить превращается в последовательность Закрыть, Применить, Закрыть.

Помимо того, что это просто глупо, это плохо уже и тем, что пользователь оказывается обманут: он-то думает, что если он нажмет кнопку Отмена, его действия в диалоговом окне не будут приняты системой во внимание. В результате, если пользователь нажмет сначала кнопку Применить, а потом кнопку Отмена, он гарантированно совершит ошибку, в которой виновата система.

Напротив, если бы вместо кнопки Применить была бы кнопка Предварительный просмотр, все бы работало великолепно. Мало того, что пользователь не путался бы в кнопках, он мог бы избежать многих ошибок, просмотрев результат своих действий перед их окончательным принятием. Но разработчикам реализовывать режим предварительного просмотра тяжело. Гораздо легче вставить кнопку Применить, а то, что пользователям это вредно, их не касается. Таким образом, кнопка Применить оказывается не просто ненужной, но и откровенно вредной. Её можно применять только в палитрах, заменяя ею кнопку ОК, чтобы показывать пользователю, что палитра не исчезнет с экрана после нажатия кнопки. Разумеется, в этом случае, с нею должна использоваться кнопка Закрыть, вместо кнопки Отмена. Во всех остальных случаях эта кнопка не нужна.

Помимо текста, на кнопках можно выводить пиктограммы. Пиктограмма (icons) – это маленькие картинки, служащие для обозначения кнопок и других объектов. Пиктограммы могут существенным образом увеличить ясность и усилить привлекательность интерфейса в данном приложении. Кроме того пиктограммы позволяют намного упростить процесс перевода названий на другие языки. Эта возможность редко используется в ПО, но очень широко в интернете. Однако у пиктограмм есть существенный недостаток – это затруднительное смысловое распознавание. В последних версиях операционных системах Macintosh и Windows при наведении курсора на пиктограмму появляется окно с текстом ее описания. Появляется естественный вопрос. Почему не использовать текст без пиктограммы? Формально, на таких кнопках пиктограммы не очень хороши из-за того, что они обычно должны передавать пользователям идею *действия* (т.е. глагол), а действие плохо передается пиктограммами. Конечно, даже и нераспознанная пиктограмма хороша тем, что она визуально отделяет кнопку от кнопки и для опытных пользователей обеспечивает ускорение при поиске нужной кнопки (пользователь может помнить, что ему нужна кнопка с синим пятном на пиктограмме). Пиктограмма будет полезной только тогда, когда она грамотно исполнена и легко читаема любым пользователем. Так что, судя по всему, пиктограммы хороши для тех кнопок, для которых пиктограммы нарисовать легко.

Для тех кнопок, которые нужны особенно часто, качество пиктограммы может особого значения не иметь, важно только различия пиктограмм между собой.

Кнопки доступа к меню

Также к группе командных кнопок относится кнопка доступа к меню. Формально, это попытка скрестить ужа (раскрывающийся список) с ежом (кнопкой), но попытка удачная. Идея проста. Существует много ситуаций, когда раскрывающийся список не помещается в отведенное для него место, поскольку текст в списке слишком велик. Первое, что приходит в голову, это вставить кнопку, нажатие на которую будет вызывать меню. В самой этой мысли нет ничего плохого, другой разговор, что без правильного исполнения она… Впрочем, по порядку.

Во-первых, недостаток кнопки будет проявляться в том, что, поскольку по условиям задачи, текст не будет виден, значение кнопки будет менее понятным, чем контекстное меню безо всякой кнопки. Формально, для совсем уж неопытных

пользователей, кнопка работать будет, но, как только пользователи подрастут, контекстное меню окажется эффективней. Как никак, в кнопку надо попасть курсором, а в меню попадать не надо, достаточно просто нажать правую кнопку мыши, огорчает также и то, что кнопки загромождают экран.

Во-вторых, само использование кнопки в таком исполнении не совсем правильно, поскольку нарушается принцип единообразия: пользователь нажал на кнопку, а действия как такового и нет (не считать же действием появление меню). В Интернете это еще проходит, поскольку там кнопки могут и не выглядеть как кнопки, будучи оформлены как ссылки; в этом случае противоречия не возникает. Суммируя, можно смело сказать, что использовать кнопку для инициирования показа меню можно, но стыдно. Не высший класс.

Существуют, впрочем, определенные ситуации, когда такие кнопки очень хороши. Для этого только нужно сделать так, чтобы кнопка была одновременно и командной кнопкой, и показывала меню. Для этого нужно сделать две вещи. Во-первых, нужно разделить кнопку на две области, одна из которых запускает действие, а другая открывает меню. Во-вторых, нужно организовать такой контекст, при котором результат нажатия на кнопку всегда будет понятным. Например, это очень хорошо работает с кнопками Вперед и Назад. Другой пример: иногда бывают ситуации, когда действий может выполняться несколько, но чаще всего нужно только одно. В этом случае пользователи очень быстро обучаются этому действию, имея довольно простой доступ к остальным. В таком исполнении кнопки доступа к меню работают замечательно.

Осталось сказать немного. Во-первых, на области, вызывающей меню, обязательно должно находиться изображение направленной вниз стрелки. Во-вторых, эта область должна находиться справа на кнопке, чтобы изображение стрелки не мешало воспринимать текст или пиктограмму на кнопке.

Чекбоксы и радиокнопки

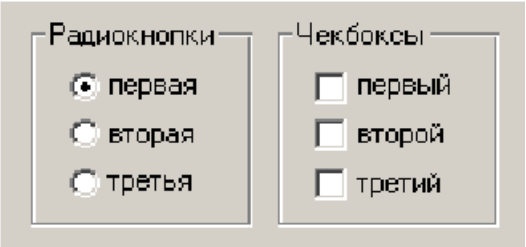

Рис. 1. Пример радиокнопок и чекбоксов.

Первое, что необходимо сказать про чекбоксы и радиокнопки, это то, что они являются кнопками отложенного действия, т.е. их нажатие не должно инициировать какое-либо немедленное действие. С их помощью пользователи вводят параметры, которые скажутся после, когда действие будет запущено иными элементами управления. Нарушать это правило опасно, поскольку это серьезно нарушит сложившуюся ментальную модель пользователей. В этом заключается общность чекбоксов и радиокнопок, теперь поговорим о различиях.

Главное различие заключается в том, что группа чекбоксов даёт возможность пользователям выбрать любую комбинацию параметров, радиокнопки же позволяют выбрать только один параметр. Это сближает эти элементы со списками множественного и единственного выбора соответственно. Из этого различия проистекают все остальные. Например, в группе не может быть меньше двух радиокнопок (как можно выбрать что-либо одно из чего-либо одного?). Еще одно следствие заключается в том, что у чекбокса есть три состояния (выбранное, не выбранное, смешанное), а у радиокнопки только два, поскольку смешанного состояния у неё быть просто не может (нельзя совместить

взаимоисключающие параметры). Но это было знание вообще. Теперь перейдем к алгоритму его использования.

В группе радиокнопок как минимум одна радиокнопка должна быть проставлена по умолчанию

Всякий раз, когда пользователю нужно предоставить выбор между несколькими параметрами, можно использовать либо чекбоксы, либо радиокнопки (или списки, но о них позже). Если параметров больше двух, выбор прост: если параметры можно комбинировать, нужно использовать чекбоксы (например, текст может быть одновременно *и* жирным *и* курсивным); если же параметры комбинировать нельзя, нужно использовать радиокнопки (например, текст может быть выровнен *или* по левому, *или* по правому краю).

Если же параметров всего два и при этом параметры невозможно комбинировать (т.е. либо ДА, либо НЕТ), решение более сложно. Дело в том, что группу из двух радиокнопок часто можно заменить одним чекбоксом. Предположим, что нужно дать пользователю выбор: показывать в документе линейки или не показывать. В этом случае логично поместить в диалоговое окно рамку группировки со словами Показывать линейки, а в эту рамку поместить две радиокнопки: Да и Нет. Понятно, что это решение очень тяжеловесно. Можно сделать проще: убрать рамку группировки и радиокнопки, а на их место поместить всего один чекбокс со словами Показывать линейки. В этом случае все будет хорошо.

К сожалению, этот метод работает не всегда. Поскольку в самом чекбоксе написано только то, что произойдет после его включения, но не описано, что произойдет, если его не включить, такая конструкция не работает в ситуациях, когда пользователям по той или иной причине функциональность непоставленного чекбокса может быть непонятна. Например, если нужно спросить пользователя, в какой кодировке посылать ему письма, не получится заменить две радиокнопки Windows 1251 и KOI-8 единым чекбоксом KOI-8. Пользователь не обязан понимать, в какой кодировке система будет посылать ему письма по умолчанию. К счастью, такие ситуации редки.

И чекбоксы и радиокнопки желательно расставлять по вертикали, поскольку это значительно ускоряет поиск нужного элемента

Внешний вид. Традиционно сложилось так, что чекбоксы выглядят как квадраты, а радиокнопки – как кружки. Нарушать это правило нельзя. Желательно вертикально располагать чекбоксы и радиокнопки в группе, поскольку это облегчает поиск конкретного элемента.

Текст подписей. Каждая подпись должна однозначно показывать эффект от выбора соответствующего элемента. Поскольку радиокнопки и чекбоксы не вызывают немедленного действия, формулировать подписи к ним лучше всего в форме существительных, хотя возможно использование глаголов (если изменяется не свойство данных, а запускается какое-либо действие). Подписи к стоящим параллельно кнопкам лучше стараться делать примерно одинаковой длины. Все подписи обязаны быть позитивными (т.е. не содержать отрицания). Повторять одни и те же слова, меняя только окончания подписей (например, «Показывать пробелы» и «Показывать табуляции»), в нескольких кнопках нельзя, в таких случаях лучше перенести повторяющееся слово в рамку группировки. Если подпись не помещается в одну строку, выравнивайте индикатор кнопки (кружок или квадрат) по первой строке подписи.

Взаимодействие. Это может показаться невероятным, но до сих пор в интернете 99% чекбоксов и радиокнопок реализованы неправильно. Дело в том, что создатели языка HTML, ничего не понимавшие в проектировании интерфейсов, были поначалу искренно уверены в том, что в этих элементах управления нажимается только визуальный индикатор переключения, т.е. кружок или квадратик. На самом деле это совершенно не так! Нажимабельной должна быть ещё и подпись, просто потому, что закон Фитса (см. «Быстрый или точный») однозначно требует больших кнопок.

Но в Интернете всего этого нет, поскольку в HTML конструкция чекбоксов и радиокнопок просто не позволяет делать нажимабельными подписи. Сейчас это стало технически возможным (через тег Label), но по инерции и вполне понятной лени никто чекбоксы нормальными не делает.

В Интернете первым признаком профессионально разработанного интерфейса являются нажимабельные подписи к чекбоксам и радиокнопкам

Другой аспект: при необходимости заблокировать элемент, желательно визуально ослаблять не только квадрат или круг, но и подпись.

Вариант для панелей инструментов

Как чекбоксы, так и радиокнопки, бывают двух видов: описанные выше стандартные, и предназначенные для размещения на панелях инструментов.

Рис. 2. Пример чекбоксов и радиокнопок на панели инструментов. Слева расположены чекбоксы (шрифт может быть и жирным и курсивным), справа радиокнопки (абзац может быть выровнен либо по левому, либо по правому краю).

Обратите внимание, что визуально чекбоксы и радиокнопки не различаются. У них есть определенный недостаток: они не различаются внешне (насколько известно, ни в одной ОС метода визуального различения не выработано). Это не очень критично, поскольку панелями инструментов пользуются в основном сравнительно опытные пользователи, так что страдать по этому поводу не стоит. Тем не менее, на панелях инструментов полезно располагать группы радиокнопок отдельно от групп чекбоксов (чтобы они не смешивались в сознании пользователей).

Вообще говоря, графические версии чекбоксов и радиокнопок можно располагать и в диалоговых окнах. Делать это, однако, не рекомендуется, поскольку в окнах они слишком уж похожи на командные кнопки, кроме того, такие кнопки не подразумевают подписей (которые в диалоговых окнах ничего не стоят, принося в то же время явную пользу). Обратите внимание, что на панелях инструментов чекбоксы и радиокнопки могут быть кнопками прямого действия.

Списки

Все часто используемые списки функционально являются вариантами чекбоксов и радиокнопок. Скорость доступа к отдельным элементам и наглядность в них принесены в жертву компактности (они экономят экранное пространство, что актуально, если количество элементов велико) и расширяемости (простота загрузки в списки динамически изменяемых элементов делает их очень удобными при разработке интерфейса, поскольку это позволяет не показывать пользователю заведомо неработающие элементы).

Списки бывают пролистываемыми и раскрывающимися, причем пролистываемые могут обеспечивать как единственный (аналогично группе радиокнопок), так и множественный выбор (чекбокс); раскрывающиеся же работают исключительно как радиокнопки. Но сначала необходимо рассказать об общих свойствах всех списков.

Ширина. Ширина списка как минимум должна быть достаточна для того, чтобы пользователь мог определить различия между элементами. В идеале, конечно, ширина всех элементов должна быть меньше ширины списка, но иногда это невозможно. В таких случаях не стоит добавлять к списку горизонтальную полосу прокрутки, лучше урезать текст элементов. Для этого нужно определить самые важные фрагменты текста (например, для URL это начало и конец строки), после чего все остальное заменить отточием (...). Поскольку нужно максимально ускорить работу пользователей, необходимо сортировать элементы. Идеальным вариантом является сортировка по типу элементов. Если же

элементы однотипны, их необходимо сортировать по алфавиту, причем списки с большим количеством элементов полезно снабжать дополнительными элементами управления, влияющими на сортировку или способ фильтрации элементов. Если можно определить наиболее популярные значения, их можно сразу расположить в начале списка, но при этом придется вставлять в список разделитель, а в систему – обработчик этого разделителя.

Пиктограммы. Уже довольно давно в ПО нет технических проблем с выводом в списках пиктограмм отдельных элементов. Однако практически никто этого не делает. Это плохо, ведь пиктограммы обеспечивают существенное повышение субъективной привлекательности интерфейса и сканируются быстрее простого текста.

Раскрывающиеся списки

Самым простым вариантом списка является раскрывающийся список. Помимо описанных выше родовых достоинств списков, раскрывающиеся списки обладают одним существенным достоинством. Оно заключается в том, что малая высота списка позволяет с большой легкостью визуально отображать команды, собираемые из составляющих.

Рис. 3. Пример визуальной сборки команды из составляющих. Данный метод значительно проще для понимания, нежели, например, ввод положительного значения для смещения вверх и отрицательного значения для смещения вниз без поддержки раскрывающимся списком. Справа на иллюстрации крутилка.

Раскрывающийся список, как правило, вызывает две проблемы, одна появляется преимущественно в ПО, другая – в Интернете. Первая проблема заключается в том, что иногда отсутствие места на экране не позволяет использовать ни чекбоксы с радиокнопками, ни пролистываемые списки множественного выбора. Приходится делать раскрывающийся список, в котором помимо собственно элементов есть «мета-элемент», включающий все элементы из списка. Этому элементу часто не дают названия, оставляя строку списка пустой, что неправильно, поскольку требует от пользователя слишком глубокого абстрагирования. Такой мета-элемент нужно снабжать названием, например, Все значения или Ничего.

В интернете есть другая проблема. Раскрывающийся список часто используется как навигационное меню. Это изначально неправильно, поскольку содержимое такого меню не видно сразу и уж тем более им трудно индицировать пользователям, в каком разделе сайта они находятся. Это, впрочем, не главное. Большая проблема заключается в том, что список снабжают скриптом, который запускается сразу по выбору значения. Такой метод имеет два недостатка. Во-первых, список исторически не является элементом управления прямого действия (как и чекбокс, например), что приводит к потере пользователями чувства контроля над системой. Во-вторых, раскрывающиеся списки довольно сложны, так что пользователи часто совершают моторные ошибки при выборе нужного элемента. Поскольку эта ошибка не может быть обнаружена системой и не всегда обнаруживается пользователями, часты ситуации, когда пользователь (как ему кажется) выбирает один раздел, а перемещается в другой, что совсем нехорошо. Таким образом, навигационные раскрывающиеся списки нужно снабжать кнопкой, которая и будет запускать действие, запускать же действие сразу после выбора элемента в списке нельзя.

Пролистываемые списки

Другим, более сложным вариантом списка является пролистываемый список. Пролистываемые списки могут позволять пользователям совершать как единственный, так и множественный выбор. Одно требование применимо к обоим типам списков, остальные применимы только к одному типу.

Размер. По-вертикали в список должно помещаться как минимум четыре строки, а лучше восемь. Напротив, список, по высоте больший, нежели высота входящих в него элементов, и соответственно, содержащий пустое место в конце, смотрится неряшливо. Требование выводить полоски прокрутки в больших списках кажется моветоном, но забывать о нем не следует.

Списки единственного выбора. Список единственного выбора является промежуточным вариантом между группой радиокнопок и раскрывающимся списком. Он меньше группы радиокнопок с аналогичным числом элементов, но больше раскрывающегося списка. Соответственно, использовать его стоит только в условиях «ленивой экономии» пространства экрана.

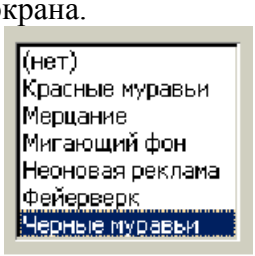

Рис. 4. Список единственного выбора. Обратите внимание, что в ситуациях, когда все элементы помещаются в список без пролистывания, список работает в точности как группа радиокнопок.

Списки множественного выбора. С точки зрения дизайна интерфейсов, списки множественного выбора интересны, прежде всего, тем, что их фактически нет в интернете. Технически создать список множественного выбора непроблематично, для этого в HTML есть даже специальный тег. Проблема в том, что такой список в браузере будет выглядеть как список единственного выбора, более того, чтобы выбрать несколько элементов пользователю придется удерживать клавишу Ctrl. Это значит, что воспользоваться таким списком сможет только малая часть аудитории (и даже наличие подсказки у списка положения не исправит). Из-за такой убогой реализации списков браузерами, использовать их, как правило, оказывается невозможно. Приходится использовать чекбоксы.

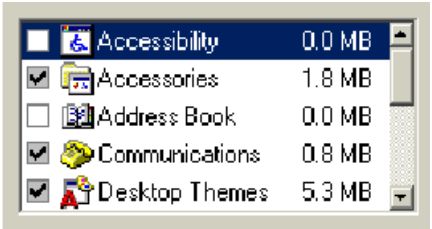

Рис. 5. Список множественного выбора с чекбоксами.

Гораздо лучше обстоят дела в ПО. Возможность безболезненно выводить в списке чекбоксы позволяет пользователям без труда пользоваться списками, а разработчикам – без труда эти списки создавать.

Комбобоксы

Комбобоксами (combo box), называются гибриды списка с полем ввода: пользователь может выбрать существующий элемент, либо ввести свой. Комбобоксы бывают двух видов: раскрывающиеся и расширенные. Оба типа имеют проблемы.

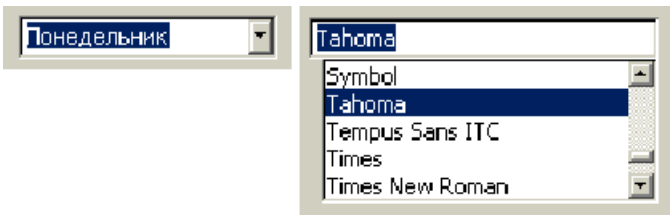

Рис. 6. Раскрывающийся комбобокс с установленным фокусом ввода (слева) и расширенный комбобокс (справа).

У раскрывающегося комбобокса есть проблемы. Во-первых, такие комбобоксы выглядят в точности как раскрывающиеся списки, визуально отличаясь от них только наличием индикатора фокуса ввода (да и то, только тогда, когда элемент выделен). Это значит, что полноценно пользоваться ими могут только сравнительно продвинутые пользователи. В этом нет особой проблемы, поскольку комбобоксом все равно можно пользоваться, как обычным списком. Во-вторых, что гораздо хуже, раскрывающиеся комбобоксы отсутствуют в интернете как класс. Поддержки их нет ни в браузерах, ни в HTML.

Проблемы расширенных комбобоксов, напротив, совершенно иные. Их с трудом, но можно реализовать в интернете (через JavaScript). Они имеют уникальный вид, отличающий их от остальных элементов управления. Зато их сравнительно трудно (хотя и гораздо легче, чем в интернете) реализовать в ПО, поскольку в Windows нет такого элемента, так что собирать его приходится из двух. При этом расширенный комбобокс потребляет много места на экране.

Поскольку комбобоксы являются гибридами списков и полей ввода, к ним применимы те же требования, что и к их родителям.

Поля ввода

Вместе с командными кнопками, чекбоксами и радиокнопками, поля ввода являются основой любого интерфейса. В результате требований к ним довольно много.

Размеры. Основная часть требований к полям ввода касается размера. Понятно, что размер по вертикали должен быть производным от размера вводимого текста – если текста много, нужно добавить несколько строк (нарушением этого правила регулярно грешат форумы, заставляющие пользователей вводить сообщения в поля ввода размером с ноготь).

С размерами по горизонтали интереснее. Конечно, ширина поля должна соответствовать объему вводимого текста, поскольку гораздо удобнее вводить текст, который *видишь*. Менее очевидным является другое соображение: ширина поля ввода не должна быть больше объема вводимого в поле текста, поскольку частично заполненное поле выглядит как минимум неряшливо.

Ширина поля ввода не должна быть больше максимальной длины строки

Отдельной проблемой является ограничение вводимого текста. С одной стороны, ограничение хорошо для базы данных. С другой стороны, всегда найдутся пользователи, для которых поле ввода с ограничением вводимых символов окажется слишком маленьким. Поэтому этот вопрос нужно решать применительно к конкретной ситуации.

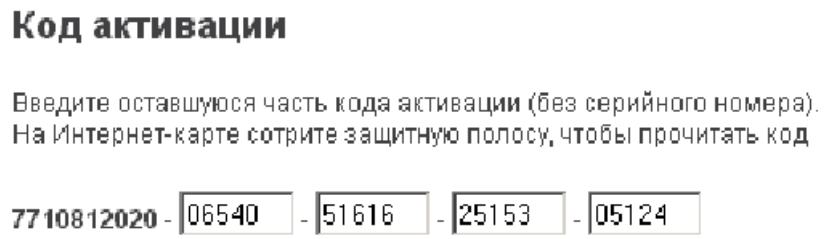

Рис. 7. Пример полей ввода, больших объема вводимых в них информации. Мало того, что такие поля выглядят неряшливо, так они ещё и обманывают пользователей, показывая, что пользователь ввел не всю информацию. Вдобавок, после заполнения поля приходится самому перемещать фокус ввода, хотя с этим справилась бы и система.

Если же суммировать информацию из двух предыдущих абзацев, можно определить самую большую ошибку, которую разработчики допускают при создании полей ввода. Всякий раз, когда ширина поля ввода больше максимального объема вводимого в него текста, при этом объем вводимого текста ограничен, пользователи неприятно изумляются, обнаружив, что они не могут ввести текст, хотя место под него на экране имеется. Соответственно, вообще нельзя делать поле ввода шире максимального объема вводимого в них текста.

Подписи. Вопрос «где надо размещать подписи к полям ввода?» является одним из самых популярных среди программистов: битвы сторонников разных подходов, хоть и бескровны, но значительны. Аргументов и подходов тут множество, но все же, поскольку восприятие подписей занимает определенное время, которого жаль, лучше всего действует следующее простое правило: в часто используемых экранах подписи должны быть сверху от поля (чтобы их было легче не читать), в редко же используемых подписи должны быть слева (чтобы всегда восприниматься и тем самым сокращать количество ошибок).

Подписи к полям ввода имеют определенное отличие от других подписей. В полях ввода подписи можно размещать не рядом с элементом, а внутри него, что позволяет экономить пространство экрана. Подпись при этом выводится в самом поле ввода, точно так же, как и текст, который в него нужно вводить. Необходимо только отслеживать фокус ввода, чтобы при установке фокуса в поле убирать подпись. Это решение, будучи нестандартным, плохо работает в ПО, но неплохо работает в интернете. Если очень жалко экранное пространство, этим методом стоит пользоваться.

Крутилки

Крутилка (spinner, little arrow) есть поле ввода, не такое универсальное, как обычное, поскольку не позволяет вводить текстовые данные, но зато обладающее двумя полезными возможностями.

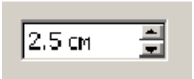

Рис. 8. Крутилка.

Во-первых, чтобы ввести значение в крутилку, пользователю не обязательно бросать мышь и переносить руку на клавиатуру (в отличие от обычного поля ввода). Поскольку перенос руки с место на место занимает сравнительно большое время (в среднем почти половину секунды), к тому же ещё и сбивает фокус внимания, отсутствие нужды в клавиатуре оказывается большим благом. Во всяком случае, ввод значения в крутилку с клавиатуры достаточно редок, т.е. пользователи воспринимают крутилки целиком и полностью положительно. Во-вторых, при вводе значения мышью система может позволить пользователям вводить только корректные данные, причем, что особенно ценно, в корректном формате. Это резко уменьшает вероятность человеческой

ошибки. Таким образом, использование крутилок для ввода любых численных значений более чем оправдано.

К сожалению, в интернете нет специального элемента для крутилки. Сделать элемент, похожий на крутилку, можно без труда, создав список множественного выбора высотой в один элемент, но ввод в него с клавиатуры будет невозможен. К счастью, крутилку можно с относительно небольшими затратами сделать в Macromedia Flash.

Ползунки

Как и ранее описанные элементы управления, ползунки позволяют пользователям выбирать значение из списка, не позволяя вводить произвольное значение. Возникает резонный вопрос: зачем нужен ещё один элемент управления, если аналогичных элементов уже полно. Ответ прост: ползунки незаменимы, если пользователям надо дать возможность выбрать значение, стоящее в хорошо ранжирующемся ряду, если:

- значений в ряду много;
- нужно передать пользователям ранжируемость значений;
- необходимо дать возможность пользователям быстро выбрать значение из большого их количества (в таких случаях ползунок оказывается самым эффективным элементом, хотя и опасен возможными человеческими ошибками).

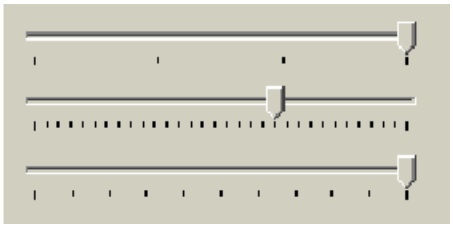

Рис. 9. Примеры ползунков. Видно, что количество параметров в ползунке может быть весьма значительным.

Ползунки имеют интересный аспект. Их можно также использовать для выбора текстовых параметров, но только в случаях, когда эти параметры можно понятным образом отранжировать. Случаев таких немало, например, «завтрак», «обед» и «ужин», при отсутствии внешней связи ранжированию поддаются вполне.

Меню

При упоминании применительно к интерфейсу термина меню, большинство людей немедленно представляют стандартные раскрывающиеся меню. В действительности, понятие меню гораздо шире. Меню – это метод взаимодействия пользователя с системой, при котором пользователь выбирает из предложенных вариантов, а не предоставляет системе свою команду. Соответственно, диалоговое окно с несколькими кнопками (и без единого поля ввода) также является меню. В настоящее время систем, которые не использовали бы меню в том или ином виде, практически не осталось. Объясняется это просто. Меню позволяет снизить нагрузку на мозги пользователей, поскольку для выбора команды не надо вспоминать, какая именно команда нужна и как именно её нужно использовать – вся (или почти вся) нужная информация уже содержится на экране. Вдобавок, поскольку меню ограничивает диапазон действий пользователей, появляется возможность в значительной мере изъять из этого диапазона ошибочные действия. Более того: меню показывает пользователям объем действий, которые они могут совершить благодаря системе, и тем самым обучают пользователей (в одном из исследований было обнаружено даже, что меню является самым эффективным средством обучения). Таким образом, в большинстве систем меню является объективным благом (они неэффективны, в основном, в системах с внешней средой или течением времени).

Типы меню

Существуют несколько различных классификаций меню, но основной интерес представляют только две из них. Первая классификация делит меню на два типа:

- Статические меню, т.е. меню, постоянно присутствующие на экране. Характерным примером такого типа меню является панель инструментов.
- Динамические меню, в которых пользователь должен вызвать меню, чтобы выбрать какой-либо элемент. Примером является обычное контекстное меню.

В некоторых ситуациях эти два типа меню могут сливаться в один: например, меню, состоящее из кнопок доступа к меню, могут работать и как статические (пользователи нажимают на кнопки) и как динамические (пользователи вызывают меню).

Вторая классификация также делит меню на два типа:

- Меню, разворачивающиеся в пространстве (например, обычное выпадающее меню). Всякий раз, когда пользователь выбирает элемент нижнего уровня, верхние элементы остаются видимыми.
- Меню, разворачивающееся во времени. При использовании таких меню элементы верхнего уровня (или, понимая шире, уже пройденные элементы) по тем или иным причинам исчезают с экрана.

Каждый тип меню в обеих классификациях имеет определенные недостатки. Статические меню, как правило, обеспечивают высокую скорость работы, лучше обучают пользователей, но зато занимают место на экране. Напротив, с динамическими меню ситуация обратная. Меню, разворачивающиеся в пространстве, обеспечивает большую поддержку контекста действий пользователей, но эта поддержка обходится в потерю экранного пространства. Меню, разворачиваемое во времени, более бережно использует пространство, но зато хуже поддерживает контекст.

Особо необходимо отметить меню типа «мастер». Являясь и динамическим и разворачивающимся во времени меню оно не оказывается более быстрым, чем, например, раскрывающееся меню. Но объем и специфика входящих в него элементов управления не позволяют, как правило, сделать из него какое-либо другое меню, например, раскрывающееся. Поэтому очень полезно научиться анализировать влияние и взаимопроникновение разных типов меню, а также осознавать их место в интерфейсе. Например, контекстное меню на ином уровне абстракции оказывается временным (т.е. динамическим) диалоговым окном, только с нестандартной структурой. Понимание этой структуры позволяет определить, какие элементы управления, помимо кнопок, можно использовать в таком меню, чтобы оно обрело как достоинства меню, так и достоинства диалогового окна.

При разработке интерфейса необходимо использовать принцип адаптивного меню. Меню делается адаптивными исходя из того предположения, что если оставлять часто используемый элемент на виду, без необходимости поиска его в меню, то это может ускорить работу пользователя.

Устройство меню

На эффективность меню наибольшее влияние оказывают устройство отдельных элементов и их группировка. Несколько менее важны другие факторы, такие как выделение элементов и стандартность меню.

Устройство отдельных элементов

Самым важным свойством хорошего элемента меню является его название. Название должно быть самым эффективным из возможного. В отличие от кнопок в диалоговых окнах, элементы главного меню практически никогда не несут на себе

контекста действий пользователя, просто потому, что в любой момент времени доступны все элементы. Это значит, что к наименованию элементов меню нужно подходить весьма тщательно, тщательней, нежели ко всему остальному. Впрочем, помимо тщательности (и таланта, к слову говоря) нужно ещё кое-что. Обязательно нужно убедиться, что выбранное название понятно целевой аудитории. Сделать это просто – пользователю нужно сообщить название элемента и попросить его сказать, что этот элемент меню делает. Нелишне заметить, что функциональность, не отраженная названием элемента, с большой степенью вероятности не будет найдена значительной частью аудитории. Поэтому не стоит умещать в диалоговое окно какую-либо функцию, если её существование в этом окне невозможно предсказать, глядя на соответствующий элемент меню. Не делайте элементов меню, часть функциональности которых не влезает в текст элемента

Особо стоит остановиться на склонении текста. В отличие от диалоговых окон, в которых кнопки прямого и отложенного действия выглядят и действуют по-разному, в меню нет четкой разницы между этими элементами. Единственным способом разграничения этих элементов является текст, так что нужно очень тщательно подходить к тому, чтобы элементы, запускающие действия, были глаголами в форме инфинитива (как командные кнопки). Впрочем, часто глагол приходится выкидывать вообще, чтобы переместить значимое слово ближе в начало текста элемента. Нужно это, чтобы повысить скорость распознавания. Повысить её можно всего одним способом: главное (т.е. наиболее значимое) слово в элементе должно стоять в элементе первым. Обратите внимание, что короткий текст элемента, без сомнения, быстро читаясь, совершенно необязательно быстро распознается. Поэтому не стоит безудержно сокращать текст элемента: выкидывать нужно все лишнее, но не более.

Пиктограммы в меню.

Пиктограммы в меню, если они повторяют пиктограммы в панели инструментов, обладают замечательной способностью обучать пользователей возможностям панели. Помимо этого они здорово ускоряют поиск известного элемента и точность его выбора, равно как и общую разборчивость меню. Таким образом, пиктограммы в меню объективно хороши (только стоят дорого, к сожалению). Это очевидный факт. Теперь менее очевидный: пиктограммы лучше работают, когда ими снабжены не все элементы. Когда все элементы имеют пиктограммы, разборчивость каждого отдельного элемента падает: в конце концов, пиктограммы всех ненужных в данное время элементов являются визуальным шумом. Когда же пиктограммами снабжены только самые важные элементы, их разборчивость повышается (а разборчивость остальных не понижается), при этом пользователям удается легче запоминать координаты элементов («элемент сразу под второй пиктограммой»).

Не снабжайте пиктограммами все элементы меню, снабжайте только самые важные

Переключаемые элементы.

Особого внимания заслуживают случаи, когда меню переключает какие-либо взаимоисключающие параметры, например, показывать или не показывать палитру. Тут есть несколько возможных способов. Можно поместить перед переключателем галочку, показывая, что он включен (если же элемент снабжен пиктограммой, можно её утапливать). Этот метод считается лучшим. Можно не помещать галочку, зато инвертировать текст элемента: например, элемент *Показывать сетку* превращается в *Не показывать сетку*. Это плохо по многим причинам. Во-первых, в интерфейсе желательно не употреблять ничего негативного: в меньшей степени потому, что негативность слегка снижает субъективное удовлетворение; в большей степени потому, что она снижает скорость распознавания текста (главное слово не первое, нужно совершить работу, чтобы из отрицания вычислить утверждение). Во-вторых, если изъять «не» и переформулировать

одно из состояний элемента, пользователям будет труднее осознать, что два разных элемента на самом деле есть один элемент. Таким образом, галочка предпочтительнее.

Всегда формулируйте текст в интерфейсе без использования отрицаний

Предсказуемость действия.

Пользователей нужно снабжать чувством контроля над системой. Применительно к меню это значит, что по виду элемента пользователи должны догадываться, что произойдет после выбора. Сделать это неимоверно трудно, поскольку на экране нет места под такие подсказки. Можно сделать только одно, но сделать это нужно обязательно: нужно показать пользователям, какой элемент запускает действие или меняет параметр, а какой открывает окно с продолжением диалога. Почти во всех ОС стандартным индикатором продолжения диалога является многоточие после текста элемента, так что пользоваться этим признаком стоит везде, включая интернет. Также необходимо показывать, какой элемент срабатывает сразу, а какой открывает элементы меню нижнего уровня (в любой ОС это делается автоматически, в Интернете нужно не забывать делать это вручную).

Это же правило касается и гипертекстовых ссылок вообще (они тоже меню). Пользователи испытывают значительно большее чувство контроля, когда имеют возможность предсказать, куда их ссылка приведет (при этом снижается количество ошибочных переходов). Таким образом, нестандартные ссылки (т.е. ссылки на другой сайт, на почтовый адрес, на файл, на узел FTP, на долго загружающуюся страницу и т.д.) полезно снабжать характерными для них признаками, например, ссылку на почтовый адрес пиктограммой письма.

Группировка элементов

Второй составляющей качества меню является группировка его элементов. В большинстве меню группировка оказывает не меньшее значение при поиске нужного элемента, нежели само название элемента, просто потому, что даже идеальное название не сработает, если элемент просто нельзя найти.

Чтобы уметь эффективно группировать элементы в меню, нужно знать ответы на три вопроса: зачем элементы в меню нужно группировать, как группировать элементы и как разделять группы между собой. Зачем элементы в меню нужно группировать. Меню, группы элементов в котором разделены, сканируется значительно быстрее обычного, поскольку в таком меню больше «точек привязки» (точно также как и в меню с пиктограммами). К тому же наличие явных разделителей многократно облегчает построение ментальной модели, поскольку не приходится гадать, как связаны между собой элементы. Наконец, в объемных меню группировка элементов облегчает создание кластеров в кратковременной памяти, благодаря чему всё меню удается пометить в кратковременную память.

Как группировать элементы.

Каждый знает, или, во всяком случае, догадывается, что элементы в меню нужно группировать максимально логично. Поспорить с этим утверждением нельзя, но от этого его проблематичность не уменьшается. Взаимоисключающие элементы желательно помещать в отдельный уровень иерархии Дело в том, что существует множество типов логики. Есть логика разработчика, который знает все функции системы. Есть логика пользователя, который знает только меньшую часть. При этом практика показывает, что эти типы логики в значительной мере не совпадают. Поскольку пользователи важнее, нужно сгруппировать меню в соответствии с их логикой. Для этого используется очень простой и надежный метод, называемый карточной сортировкой.

Как разделять группы между собой.

Существует два основных способа разделять группы: между группами можно помещать пустой элемент (разделитель) или же размещать отдельные группы в разных уровнях иерархии. Второй способ создает более четкое разделение: в меню Файл, например все элементы более близки друг другу (несмотря на разделители), чем элементы других меню. В то же время выбор конкретного способа диктуется результатами карточной сортировки, так что интерес представляет только вопрос «как должны выглядеть и действовать разделители».

Для разграничения групп традиционно используют полоски. Это надежное, простое решение, другой разговор, что с дизайнерской точки зрения полоски плохи, поскольку представляют собой визуальный шум. Гораздо правильнее, но и труднее, использовать только визуальные паузы между группами, как это сделано, например, в MacOS X.

Глубина меню.

Наличие многих уровней вложенности в меню приводит к там называемым «каскадным ошибкам»: выбор неправильного элемента верхнего уровня неизбежно приводит к тому, что все следующие элементы также выбираются неправильно. При этом широкие меню больше нравятся пользователям. Поэтому большинство разработчиков интерфейсов стараются создавать широкие, а не глубокие меню.

К сожалению, у широких меню есть недостаток: они занимают много места. Это значит, что, начиная с определенного количества элементов, меню физически не сможет оставаться широким, оно начнет расти в глубину. Возникает проблема, которую надо решать. Итак, проблема заключается в том, что велика вероятность каскадных ошибок. Чтобы снизить их число, нужно повысить вероятность того, что пользователи будут правильно выбирать элементы верхних уровней. Чтобы повысить эту вероятность, нужно заранее снабдить пользователей контекстом. При перемещении по меню пользователь действует по определенному алгоритму:

1 Выбирая элемент первого уровня, он выбирает элемент, «нужность» которого кажется ему максимальной.

2 После выбора он видит список элементов второго уровня, при этом он оценивает вероятность соответствия всех элементов второго уровня его задаче и одновременно выбирает наиболее вероятный элемент. При этом в уме он держит контекст, т.е. название элемента первого уровня.

3 Если ни один из элементов не кажется пользователю достаточно вероятным, пользователь возвращается на первый уровень.

4 Если какой-то элемент удовлетворяет пользователя, он выбирает его и получает список элементов третьего уровня. Действия из второго и третьего шагов повторяются применительно к новым элементам меню.

Видно, что действия пользователя при поиске нужного элемента отчетливо цикличны, при этом на каждом шаге есть вероятность ошибок. При этом с каждым новым уровнем меню объем контекста, который приходится держать в голове, непрерывно возрастает. При этом, если пользователь всё-таки не находит нужного элемента, весь этот контекст оказывается ненужным. Хранение же контекста, даже не засчитывая усилия, затрачиваемые на выбор элемента, есть довольно существенная работа. Её объем лучше уменьшить.

Теперь рассмотрим другой вариант: пользователь по самому элементу может предугадать его содержимое, т.е. при поиске элемента в меню не столько оценивает контекст, сколько просто ищет нужный элемент. Эта возможность есть в любом случае, поскольку элемент имеет хоть сколько-нибудь значимый идентификатор (т.е. его название). Но она, как правило, довольно слаба и почти всегда допускает неоднозначность. Усилить её можно наличием аннотации к каждому элементу, но эту аннотацию никто не будет читать.

Есть другой метод, и этот метод есть, пожалуй, лучшее, что дал Интернет науке о проектировании интерфейсов: в качестве аннотации к элементу можно показывать наиболее популярные элементы следующего уровня. В этом случае пользователь может сформировать контекст элемента, не перемещаясь внутрь этого элемента, при этом вероятность ошибочного перехода значительно снижается. Помимо уменьшения числа ошибок, такая система позволяет ускорить доступ к наиболее популярным элементам второго и последующих уровней.

В целом, ширина и глубина меню являются, пожалуй, наименее значимыми факторами. Гораздо важнее хорошая группировка, при этом как группировку, так и структуру дерева меню, всё равно лучше определять карточной сортировкой. Применительно же к раскрывающимся меню действует ещё один ограничитель глубины. Раскрывающиеся меню довольно тяжелы в использовании, поскольку требуют от пользователей достаточно тонкой моторики. Поэтому главное меню с уровнем вложенности элементов большим трех просто невозможно.

Контекстные меню

Преимущество контекстных (всплывающих) меню заключается в том, что они полностью встраиваются в контекст действий пользователей: не нужно переводить взгляд и курсор в другую область экрана, практически не нужно прерывать текущее действие для выбора команды. При этом они не занимают места на экране, что всегда ценно. С другой стороны, из-за того, что они не находятся всё время на экране, они практически неспособны чему-либо научить пользователя.

Не делайте контекстные меню единственным способом вызова какой-либо функции

Поскольку основной причиной появления контекстных меню является стремление максимально повысить скорость работы пользователей, на их размер и степень иерархичности накладываются определенные ограничения. Если меню будет длинным, пользователям придется сравнительно долго возвращать курсор на прежнее место, так что привлекательность нижних элементов окажется под вопросом. Поэтому лучше сокращать размер контекстных меню до разумного минимума (порядка семи элементов). К тому же не надо забывать, что главное меню не всегда перекрывает выделенный (т.е. актуальный объект), а контекстное меню – почти всегда (как-никак оно вызывается на самом объекте). В большинстве же случаев перекрытие актуального объекта нежелательно (сбивается контекст). Мы не можем сделать в этой ситуации ничего, кроме как уменьшить размер меню, в расчете, что маленькое меню будет перекрывать малое количество информации. Разумеется, если точно известно, что оперируемый объект совсем уж мал, сокращать объем меню бесполезно.

Другая особенность контекстных меню – иерархия. В обычном меню иерархия имеет хотя бы одно достоинство: при обучении она позволяет упорядочивать элементы меню и тем самым делать его понятнее. В контекстных же меню обучающая функция не играет никакой роли, поскольку такими меню пользуются только опытные пользователи. Иерархия элементов теряет свое единственное достоинство, не теряя ни одного недостатка. Поэтому делать иерархические контекстные меню можно, ничего плохого в этом нет, но необходимо сознавать, что вложенными элементами почти никто не будет пользоваться (тем более что вложенность сбивает контекст действий).

Система сначала должна показывать максимально релевантную (уместную) информацию, затем всё остальное. Последнее отличие контекстных меню от обычных заключается в том, что в них очень важен порядок следования элементов. В главном меню не обязательно стремиться к тому, чтобы наиболее часто используемые элементы были самим первыми – все равно курсор придется возвращать к рабочему объекту, так что

разницы в дистанции перемещения курсора практически нет. В контекстном же меню ситуация обратная – чем дальше нужный элемент от верха меню, тем больше придется двигать курсор. Поэтому правило релевантности (уместности) в таких меню действует в полной мере.

Окна

Поскольку разработка интерфейса заключается в основном в том, чтобы правильно помещать правильные элементы управления в правильные окна или экраны, окна требуют не меньше заботы, чем элементы управления.

Типы окон

Современная наука знает несколько типов окон, а именно:

- главные окна программы
- окна документа
- режимные диалоговые окна
- безрежимные диалоговые окна
- палитры
- окна браузера (поскольку используемая в интернете технология существенно отличается от технологии ПО, этот тип окон стоит несколько особняком).

При этом доля отдельных типов в общем пироге со временем изменяется: окна документов, как будет показано ниже, отмирают, заменяясь окнами программ, режимные диалоговые окна сменяются безрежимными, а безрежимные, в свою очередь, палитрами. Интересно, что идея палитр тоже клонится к закату (палитры сменяются панелями инструментов, причины этого рассмотрены ниже), так что в будущем, скорее всего, в ПО останутся только окна программ, панели инструментов и безрежимные диалоговые окна (которые разработчики поленятся переделывать). Но об этом отдельно.

Недолгая история окон на экране

Сейчас многим в это трудно поверить, но сравнительно недавно никаких окон не было, даже диалоговых окон, которые уже стали восприниматься как данность. Вместо них какая-то часть экрана выделялась под меню, которое в те времена было функционально более богатым, чем меню теперешнее (так, нормой были поля ввода в меню).

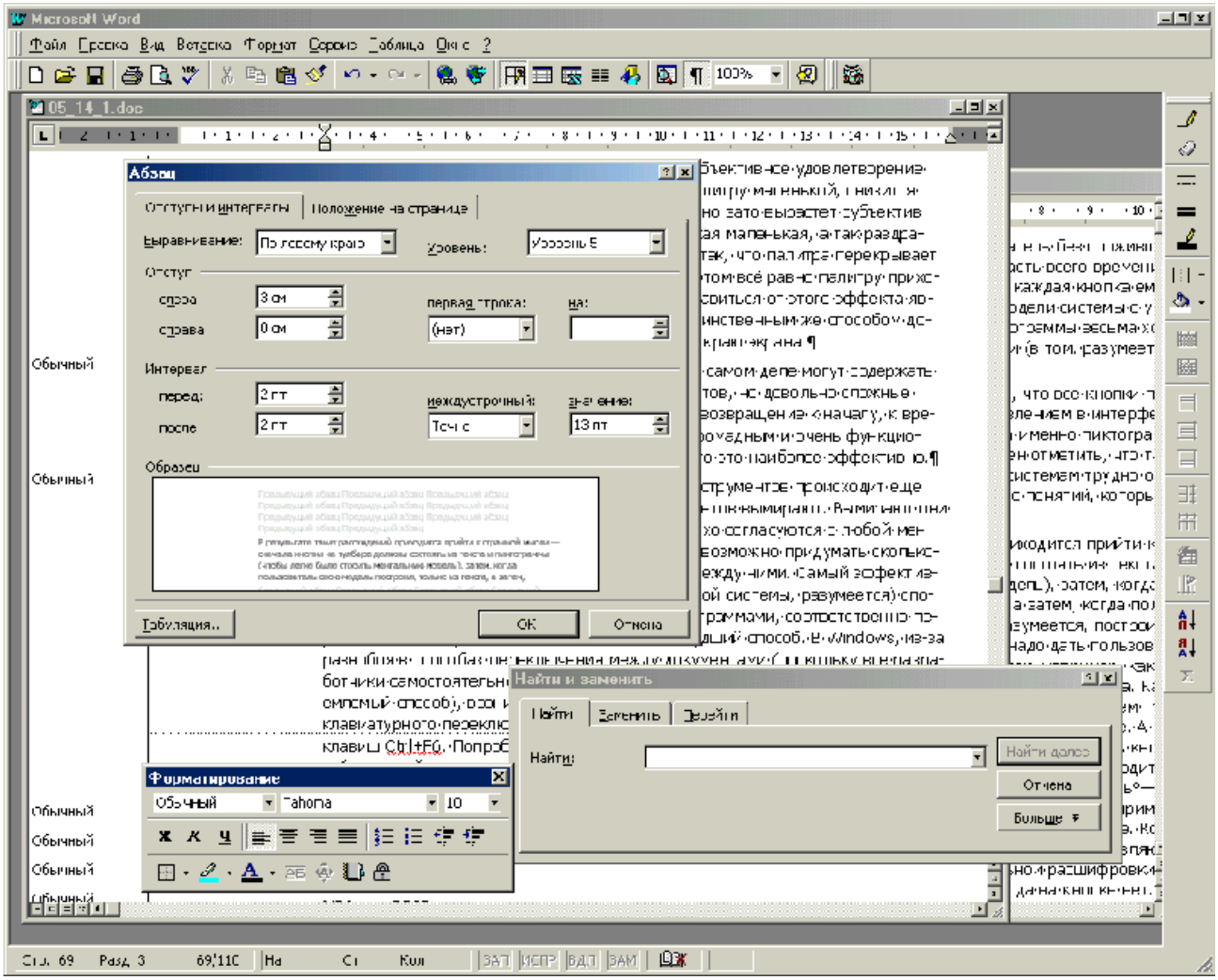

Рис. 1. Типы окон на примере MS Word 97.
Самое большое окно есть окно программы. Внутри него два окна документов, в более свежих версиях Word их уже нет. Слева сверху располагается режимное диалоговое окно (Абзац), под ним справа – безрежимное (Найти и заменить). Слева внизу располагается палитра. Сверху и справа две панели инструментов (бывшие палитры).

Потом, с появлением графического режима, стало возможным реализовать в интерфейсе метафору рабочего стола (если бумажные документы могут лежать на столе друг на друге, то почему этого не могут делать электронные документы?). Появились окна программ, окна документов и диалоговые окна, первоначально сплошь режимные.

Понятие «режимное диалоговое окно» кажется довольно загадочным (еще более загадочным кажется его англизированный вариант «модальное диалоговое окно»). На самом деле всё просто. Если открывшееся окно блокирует доступ к остальной части системы, происходит, фактически, запуск нового *режима работы* (поскольку функциональность отдельного диалогового окна никогда не совпадает с функциональностью системы в целом). После того, как окно закрыто, происходит возвращение предыдущего (основного) режима. В этом и есть всё значение термина «режимный».

Прошло несколько лет, и наличие режима в диалоговых окнах стало немодным. Во-первых, всех раздражает, что, вызвав диалоговое окно и обнаружив, что вызвано оно преждевременно, приходится закрывать окно и открывать его в следующий раз заново. Во-вторых, что важнее, в системах, ориентированных на документы, режим сбивает

внимание пользователя и вообще лишает его ощущения управляемости (в отличии систем, ориентированных на формы ввода, в которых режим работает лучше, чем его отсутствие). В-третьих, сама по себе идея сближения интерфейса с реальным миром (в частности, метафора рабочего стола) протестовала против идеи режимов в любом их проявлении, поскольку в реальном мире вообще не бывает режимов, аналогичным интерфейсным. А поскольку «дизайн пользователей» был ориентирован на функционирование в реальном мире, решили не переделывать пользователей, а переделать интерфейс.

Избегайте режимов работы

Так появились безрежимные диалоговые окна, т.е. окна, которые можно было неограниченное время держать на экране, переключаясь по мере надобности между ними и собственно документом. К сожалению, и здесь не без проблем. Дело в том, что такие диалоговые окна нельзя делать тонущими, т.е. позволять пользователю перекрывать их окнами документа или программы. Причина проста – пользователи забывают, что они когда-то открывали соответствующее окно и пытаются открыть его заново. Зачем, спрашивается, такие окна? Поэтому решили сделать такие окна плавающими, т.е. перекрываемые только другими плавающими окнами этой же программы или другими программами. Разумеется, некоторые диалоговые окна невозможно сделать безрежимными: например, что делать с сообщениями об ошибках? Но, в целом, с переводом окна в безрежимное состояние нет особой проблемы.

Но и тут обнаружилась проблема. Дело в том, что просто диалоговое окно, даже будучи безрежимным, малополезно, поскольку перекрывает слишком много важного и нужного. Решение этой проблемы было эволюционным, и поэтому относительно простым – были придуманы палитры, т.е. окна, из которых выжали всё пустое место. Сразу оказалось, что палитры, помимо малых размеров, имеют одно большое достоинство: пользователи очень любят их расставлять на экране индивидуальным порядком. Пользы это особой не приносит, зато существенно повышает субъективное ощущение контроля над системой. К сожалению, визуальный дизайн палитр, как правило, довольно сложен и длителен, так что сугубо экономические причины мешают переделать в палитры все диалоговые окна.

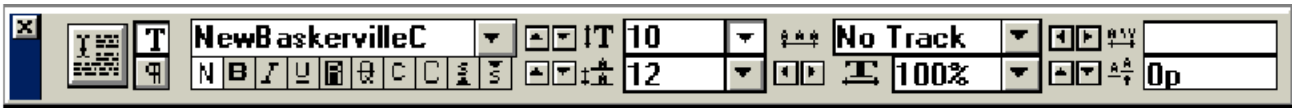

Рис. 2. Пример палитры из программы Adobe PageMaker.

Как легко догадаться, проблема была найдена и в палитрах. Существует неформальный, но на удивление верный закон, гласящий, что субъективная важность информации, перекрываемой диалоговым окном (палитрой в частности), не зависит ни от размеров, ни от положения окна, а зависит только от периметра. В результате постоянно оказывается, что пользователи, стараясь открыть нужную информацию, перекладывают окна с места на место, что снижает производительность (несущественно) и субъективное удовлетворение (существенно). При этом если сделать палитру маленькой, снизится вероятность её вынужденного перетаскивания, но зато вырастет субъективное неудовольствие от её перетаскивания («такая маленькая, а так раздражает»). Более того. Гораздо чаще оказывается так, что палитра перекрывает не всю нужную информацию, но её часть; при этом всё равно палитру приходится перемещать. Единственным способом избавиться от этого эффекта является уменьшение периметра палитры, а добиться этого можно, только прикрепив палитры к краю экрана.

Так родились панели инструментов, которые на самом деле могут содержать (и содержат) не только пиктограммы инструментов, но довольно сложные элементы

управления. В некотором смысле, это возвращение к началу, к временам, когда один из краев экрана был занят громадным и очень функциональным меню. С другой стороны, оказалось, что это наиболее эффективно.

Параллельно с рождением сложных панелей инструментов происходит ещё одна драма борьбы за выживание. Окна документов вымирают. Вымирают они по двум простым причинам. Во-первых, они плохо согласуются с ментальной моделью большинства пользователей. Во-вторых, невозможно придумать сколько-нибудь эффективного способа переключаться между ними. Самый эффективный (с точки зрения разработчиков операционной системы, разумеется) способ обычно отдается переключению между программами, соответственно, переключению документов достается заведомо худший способ. В Windows, из-за разнобоя в способах переключения между документами (поскольку все разработчики самостоятельно старались найти какой-либо приемлемый или неприемлемый способ), возникают казусы: в MS Word, например, для клавиатурного переключения между документами используется комбинация клавиш Ctrl+F6. Попробуйте использовать эту комбинацию клавиш одной рукой, и вы поймете, что это невозможно.

Для того чтобы запустить две одинаковые программы, каждая с одним документом внутри, не хватало ресурсов компьютера, вот и приходилось запускать одну программу с двумя документами. Сейчас, напротив, памяти достаточно, к тому же появились технологии программирования, позволяющие ни о чем таком даже не думать. Так что окна документов постепенно становятся ненужными.

Элементы окна

Окна, помимо областей с элементами управления, имеют некоторые общие элементы, главными из которых являются строки заголовка окна, строки статуса, панели инструментов и полосы прокрутки.

Строка заголовка окна

У каждого окна есть строка заголовка. Поэтому пользователи строкой заголовка интересуются весьма мало, можно сказать, совсем не интересуются. Человеку не свойственно обращать внимание на обыденность, особенно если эта обыденность не находится в фокусе его внимания (а строка заголовка как раз в нем не находится). В результате, пользователи обращают внимание на строку заголовка, только обучаясь пользоваться компьютером или в ситуациях, когда они совсем ничего не понимают в системе. Из этого, однако, не следует, что строкой состояния можно пренебрегать. Точнее, самой строкой как раз пренебречь можно, но её содержимым – нельзя.

Дело в том, что текст и, в меньшей степени, пиктограмма заголовка играют важную роль в ПО (они заведуют переключением задач) и очень важную в интернете (заведуют навигацией). Панель задач в Windows создает по кнопке для каждой запущенной программы. Поскольку ширина экрана ограничена, при увеличении количества запущенных программ размеры кнопок сокращаются, соответственно в эти кнопки помещается меньше текста. В результате пользователь сохраняет способность опознать программу по её пиктограмме и обрывку текста, но теряет возможность опознавать документы. Проблемы можно было бы избежать, если бы название программы на кнопке (и в строке заголовка окна) было бы короче и/или название документа выводилось бы *до* названия программы. Заодно пользователям не пришлось бы сотни и тысячи раз читать название программы (последствия неумеренного продвижения торговой марки). С переключением задач всё просто и сложно одновременно. Просто, поскольку правило тут простое «Релевантное выводится в первую очередь». Поскольку пользователю нужен именно конкретный документ конкретной программы, а вовсе не программа просто (мы уже определили, что окна документов, не попадающие в переключатель задач, нехороши), названия документов, как более релевантные, нужно

выводить в первую очередь. Наоборот, сложность состоит в том, что из-за жесткости интерфейса Windows много не сделаешь. Тем не менее, сокращать название программы нужно безусловно.

Иная ситуация в интернете. Поскольку пиктограмма в строке заголовка приходит от браузера, нет особой возможности оптимизировать переключение задач. С другой стороны, качество этого заголовка оказывает существенное влияние на навигацию, поскольку при показе результатов поиска в поисковых системах заголовком элемента становится содержимое тега Title. Каковое содержимое и попадает в обычном режиме наверх экрана. При этом в интернете нет проблемы с текстом заголовка – что хотим, то и пишем (стараясь не обращать внимания на то, что к этому прибавится название браузера).

Нажатие на пиктограмму в строке заголовка вызывает раскрывающееся меню, являющееся замечательным местом для вызова функций, которые нужны только наиболее опытной аудитории

Правило релевантности действует и здесь – в начале строки должна быть более релевантная информация, нежели в её конце. Поскольку связки «программа-документ» в интернете нет, эффективнее всего показывать адрес текущей страницы в навигационной системе сайта (если сайт иерархический). В данном случае релевантность требует, чтобы сначала шло название текущего документа, затем раздела, в котором он находится, затем раздела более высокого уровня и так далее. Не надо также забывать, что размер строки ограничен, так что более 70-80 символов в ней быть не может.

Также важно понимать, что тот факт, что пользователи редко читают заголовки окна, вовсе не означает, что заголовки пользователям не нужны. Напротив, хороший заголовок может здорово облегчить понимание работы диалога. Поэтому наличие на экране заметного и адекватного заголовка окна часто оказывается очень полезным. Жалко только, что в обычном Windows-интерфейсе места под него нет.

Панели инструментов

Все панели имеют следующие достоинства:

- они позволяют пользователям быстро вызывать нужные функции мышью
- они позволяют пользователям меньше задействовать память
- они повышают визуальное богатство интерфейса
- они ускоряют обучение работе с системой (по сравнению с раскрывающимся меню) благодаря своей большей наглядности.

Зато они имеют и недостаток: занимают много места на экране, так что поместить в них всё, что хочется, невозможно. Решить эту проблему можно двояко. Во-первых, можно (и нужно) помещать в панель только наиболее часто используемые команды (поддерживая это решение возможностью индивидуальной настройки панели пользователем). Во-вторых, панель можно сделать зависимой от контекста действий пользователя. Оба способа не противоречат друг другу, так что использовать стоит оба.

Панель инструментов нежелательно делать единственным способом вызова функции

В настоящее время нет технической проблемы с помещением в панели произвольных элементов управления (остался только один ограничитель – размер помещаемых элементов), так что последние преграды, мешавшие делать сложные панели, исчезли. Этим стоит пользоваться, поскольку это позволяет экономить время, уходящее на открытие и закрытие диалоговых окон, и повышать интегральное качество взаимодействия с системой (пользователям *нравится* пользоваться сложными панелями).

Текст на кнопках. Самыми частыми элементами управления, размещаемыми на панелях инструментов, является командные кнопки, при этом их использование отличается от обычного. Дело в том, что места настолько не хватает, что очень хочется заменить текст кнопок пиктограммами. Но это не так просто.

Дело в том, что когда приходит время совершить выбор, имея в качестве альтернатив визуальные объекты, «человек выбирающий» чаще всего транслирует эти объекты в звуки, а именно в слова (в голове, разумеется). Затем эти слова помещаются в кратковременную память, в дело включается собственно сознание (предыдущие этапы проходят на бессознательном уровне) и выбирает нужный объект. Применительно к реальной жизни это значит, что пользователь, глядя на панель с пиктограммами, видит скорее не пиктограммы, но слова. Но не всегда.

Случай 1. Опытный пользователь, уже знающий, где на панели находится нужная кнопка, знающий её значение, при этом выбор действия уже произведён при помощи сложившейся ментальной модели. В такой ситуации слова пользователю уже не важны, важно отличие нужной ему кнопки от остальных. Т.е. такому пользователю даже уже все равно, что на пиктограмме изображено, лишь бы она выглядела максимально контрастно (чтобы ускорить её поиск).

Случай 2. Опытный пользователь, обладающий сложившейся ментальной моделью, но не знающий, где конкретно расположена нужная ему кнопка и как она выглядит. Выбор действия уже произведен, осталось только найти нужную кнопку. При этом пиктограмма оказывается ненужной, так как в качестве матрицы пользователь использовать её не может (поскольку не знает, как она выглядит). Более того, поскольку пользователь ищет слово из содержимого своей кратковременной памяти, каждая пиктограмма будет его без пользы отвлекать, при этом пользователь будет тратить время на расшифровку смысла всех попадающихся ему на пути пиктограмм.

Случай 3. Неопытный пользователь без сложившейся ментальной модели. Такой пользователь большую часть всего времени тратит на поиск нужной ему кнопки, а также, поскольку каждая кнопка ему внове, на постоянное улучшение своей ментальной модели системы с учетом своих новых открытий. В таких случаях пиктограммы лучше текста, но не заменяют его, так как помогают *быстрее* понять действие кнопки (в том, разумеется, случае, когда пиктограмма адекватна смыслу действия).

В результате таких рассуждений приходится прийти к странной мысли – сначала кнопки на панели инструментов должны состоять из текста и пиктограммы (чтобы легко было строить ментальную модель), затем, когда пользователь свою модель построил, только из текста, а затем, когда пользователь окончательно обучился пользоваться системой, только из пиктограммы. Разумеется, построить такую систему невозможно, так что приходится определяться. Поскольку в двух случаях из трех текст оказывается нужен (тем более что начинающие и средне продвинутые пользователи составляют большинство), удалять его из панели оказывается неправомерным.

Здесь действует ещё один закон. Поскольку кнопка с пиктограммой и текстом всегда больше кнопки с текстом или пиктограммой просто, она оказывается более эффективной в отношении скорости, поскольку в неё легче попасть мышью. Таким образом, эффективнее всего (учитывая все аргументы за и против) делать кнопки на панелях инструментов диалектически: самые главные кнопки нужно делать парой «пиктограмма плюс текст», а остальные в зависимости от их направленности – функции для опытных пользователей пиктограммами, а для неопытных текстом.

Полосы прокрутки и их альтернатива

Когда графических интерфейсов еще не было, пользователи перемещались по документу с помощью клавиатуры. С тех далёких времен на клавиатуре остались клавиши Home и End, равно как Page Up и Page Down. В целом, пользователи были удовлетворены

своей судьбой. Затем появились графические интерфейсы. Первым делом были придуманы полосы прокрутки. К сожалению, оказалось, что они работают не слишком хорошо.

Проблема полос прокрутки заключается в следующем: для маленьких документов они не очень нужны, поскольку пользователям, держащим руки на клавиатуре, гораздо легче переместиться к нужному фрагменту с помощью клавиш со стрелками. Напротив, в больших документах малое перемещение ползунка приводит к существенному сдвигу области просмотра, так что после перемещения нужно еще и подправляться либо клавиатурой, либо стрелками на полосе прокрутки. Более того: во многих случаях невозможно реализовать динамическое изменение области просмотра во время перемещения ползунка, а значит, перемещаться по большим документам приходится в несколько шагов. И еще раз более того: предположим, что это динамическое изменение всё-таки есть. Тогда пользователю нужно: сначала перевести взгляд на ползунок, затем курсор на ползунок, затем взгляд на документ и только потом, перемещая мышь вверх или вниз, следить за областью просмотра, на тему «не пора ли отпустить курсор».

Пользователям не нравятся горизонтальные полосы прокрутки

Нечего и говорить, что пользователи избегают пользоваться полосками прокрутки (тем более что курсорные клавиши никто с клавиатуры не убирал). Фактически, чуть ли не единственным применением этих полосок является перемещение «примерно к нужному фрагменту» при работе с большими документами.

Разумеется, такое положение вещей никого особенно не радовало. Поэтому было придумана «дополнительная стоимость» полосок – размер ползунка был сделан пропорциональным отношению видимой части документа ко всему его объёму. Это очень хорошая и полезная функция, поскольку она позволяет использовать полосы прокрутки не только как элемент управления, но и как индикатор размера документа, благодаря чему степень бесполезности полосок значительно снижается. Помимо этого было придумано довольно много других дополнительных стоимостей, так, например, на полоске прокрутки можно отображать границы разделов документа.

Полосы прокрутки без индикации размера документа практически бесполезны

Тем не менее, всё это так и не сделало полосы прокрутки здорово лучше: как и раньше, полосы не столько помогают перемещаться по документу, сколько показывают то, что не весь документ помещается на экране. Решение этой проблемы пришло с несколько непривычной стороны, во всяком случае, графический пользовательский интерфейс не пригодился – была придумана мышь с колесиком прокрутки. Решение это чуть ли не идеальное, поскольку не требует от пользователя переносить внимание с документа на элемент управления. Конечно, для перемещения по большим документам колесо не слишком эффективно (палец устаёт), но малые и средние перемещения получаются замечательно, тем более что процент больших документов невелик. Поскольку мышь стоит не слишком дорого, а служит не слишком долго, сейчас можно смело рассчитывать на наличие у пользователей мышей с колесиком.

Таким образом, полосы прокрутки стали ёще более бесполезны, поэтому относиться к ним надо не как к данности, но как к еще одному элементу управления, который можно использовать, а можно и не использовать. При этом есть как аргументы в пользу использования, так и существенный аргумент против него – полоски прокрутки занимают много места на экране. Ладно еще, когда на экране одна полоска, а что делать, если их три или более?

С появлением мышей с колёсиками, полоски прокрутки смело можно делать тоньше

К сожалению, вовсе не использовать полосы прокрутки в ПО затруднительно, MS Windows User Experience прямо заставляет разработчика ими пользоваться. В интернете ситуация иная – никто никого не заставляет. Осталось разобраться, как же сделать пролистывание документа идеальным.

Если всё-таки приходится оставлять полосы прокрутки, крайне желательно добиться нескольких свойств полосок:

- Размер ползунка должен показывать общий объем пролистываемого документа.
- Стрелки на полосах должны быть спаренными, т.е. обе стрелки должны находиться рядом, а не на разных сторонах полоски. Это один из случаев, когда логичность интерфейса вступает в противоречие с эффективностью. Если при перелистывании была допущена ошибка, спаренные кнопки позволяют минимизировать перемещение курсора к стрелке, ведущей в обратную сторону.
- Если невозможно сделать динамическое изменение области просмотра при пролистывании, необходимо показывать текущее местоположение пользователя во всплывающей подсказке.
- Необходимо обеспечить обработку погрешности перемещения курсора. Когда пользователь курсором перемещает ползунок, а смотрит в это время на документ, курсор может сойти с полосы. До определённого момента (смещение на 30-70 пикселей) система должна такое смещение игнорировать.

Теперь об альтернативных элементах управления. Чаще всего используются кнопки со стрелками, т.е. фактически полоски прокрутки, из которых вырезано самое главное. Это не очень хороший элемент, потому что он совершенно линеен: когда пользователь нажимает на кнопку со стрелкой, документ листается с фиксированной скоростью, изменить которую пользователь не в силах. Это приводит либо к медленному пролистыванию, либо к низкой точности. Поэтому гораздо эффективнее малюсенький джойстик, часто встречающийся в ноутбуках. Сущность этого элемента проста: на экране располагается объект, нажатие на который меняет курсор на изображение направленных в разные стороны стрелок. Если пользователь перемещает курсор с нажатой кнопкой мыши в сторону, документ в эту же сторону и прокручивается, причем скорость прокрутки пропорциональна расстоянию, на которое перемещен курсор. Важно только не забывать его блокировать, когда пролистывать нечего. Такой элемент управления в настоящее время реализован в MS Windows и доступен по нажатию средней кнопки мыши. Структура окна Структура и само устройство окна или экрана является, пожалуй, самым существенным фактором, влияющим на качество интерфейса в этом окне. Например, производительность пользователей порой можно повысить вдвое, просто изменив расположение элементов управления, не меняя сами эти элементы.

Большинство руководств по проектированию интерфейсов, перечисляя требования к структуре окна, ограничиваются замечанием, что терминационные кнопки (т.е. командные кнопки, управляющие окном, например Ok или Закрыть) должны быть либо снизу окна, либо в правой его части. Это хорошо, но мало. На самом деле всё сложнее. Во-первых, окно должно хорошо сканироваться взглядом, т.е. его основные части должны быть сразу видны и заметны. Как правило, в окнах с малым количеством элементов управления проблем со сканированием не возникает. Проблемы появляются в больших окнах, дающих доступ ко многим функциям. Понятно, что сваливать эти функции в кучу неэффективно, для этого интерфейсные элементы должны быть организованы в блоки. В ПО для этого используются в основном рамки группировки, в интернете – пустоты, разграничивающие отдельные функции. При этом рамки удобнее в производстве, но, поскольку они являются визуальным шумом, однозначно рекомендовать их нельзя. В целом, разграничивать блоки пустотами предпочтительней (но и сложней).

Во-вторых, окно должно *читаться*, как текст. При прочих равных, окно, все элементы управления которого можно без труда связно прочесть, будет лучше запомнено и быстрее обработано мозгом (поскольку не придется преобразовывать грамматику окна в грамматику языка).

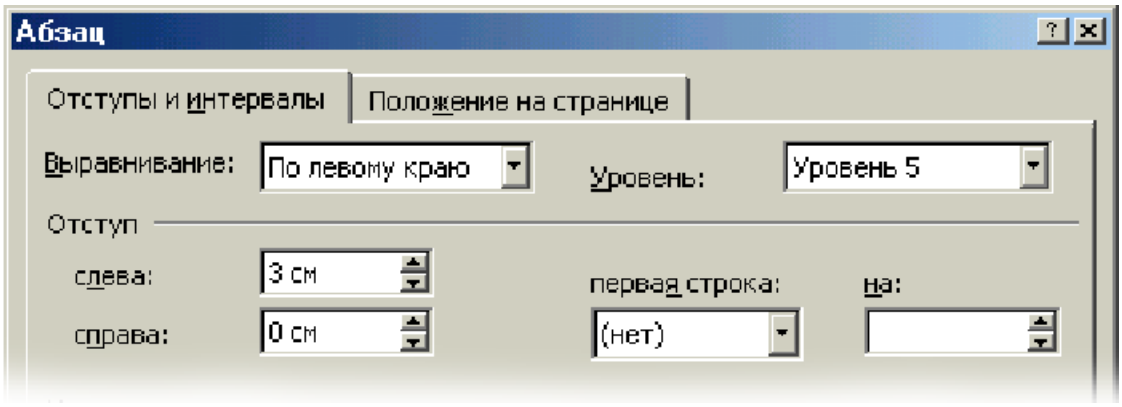

Рис. 3. Пример читаемого окна.

Читается он следующим образом: «Текст выравнивается по левому краю, уровень пятый, отступ слева 3 см, справа 0 см, первая строка нет, на 5 и так далее».

На этом примере прекрасно видны все неопределенности в окне: например, не говоря уже о том, что непонятно, чего именно пятый уровень, видно, что подписи к «первая строка» и к «на», расположенные сверху, разрывают единый по смыслу элемент управления на два разных. При этом один элемент управления должен однозначно преобразовываться в единый фрагмент предложения, а единая группа элементов – в целое предложение.

Окно должно читаться, как текст

Проверить читаемость окна исключительно просто: окно нужно просто прочитать. При этом становится понятно, какие нужны подписи к элементам, как они должны быть расположены и тому подобное. В-третьих, оно должно удовлетворять закону «релевантное – вперед». Чаще всего используемые элементы должны быть расположены в левой верхней части экрана, реже используемые – в правой нижней части. Обратите внимание, что окно сканируется взглядом точно так же, как происходит процесс чтения – сначала взгляд переходит в левый верхний угол, затем перемещается вправо, затем переходит на следующую «строку» и т.д. Поэтому, например, вертикальный элемент управления, разрывающий эти воображаемые строки на части, будет всегда замедлять сканирование окна (и вызывать неудовольствие у пользователей).

Теперь, возвращаясь к началу, пора объяснить, почему терминационные кнопки должны быть расположены внизу или справа, тем более что здесь действует всеобъемлющий закон. Дело в том, что в интерфейсе всегда должно быть реализовано правило: сначала выбор параметров, а затем действие (интересно, что в большинстве языков ситуация обратная, например, мы не говорим «Петю укусить», но говорим «укусить Петю»). Нарушение этого правила существенно повышает количество человеческих ошибок и ослабляет пользовательское ощущение контроля (что грозит низким субъективным удовлетворением). Это правило, будучи применено к диалоговым окнам, и заставляет помещать терминационные кнопки снизу или справа, т.е. в области, которая сканируется последней.

Увеличиваем площадь

Площадь экрана ограничена, напротив, количество элементов управления, которых может понадобиться уместить в едином функциональном блоке (т.е. окне), не ограничено ничем. В этом случае приходится использовать вкладки (см. рис. 5.2). Чтобы правильно их использовать, нужно соблюдать определенные требования.

Первая вкладка и остальные вкладки. Помимо умещения максимального количества элементов управления в диалоговом окне, вкладки могут выполнять еще одну роль, а именно скрывать от неопытных пользователей не очень нужную им функциональность. Проблема заключается в том, что когда нужно просто уместить в окно побольше элементов, вкладки скрывают от пользователей функциональность, возможно, что и нужную. Некогда в Windows было два способа поместить в диалоговое окно больше, чем в него могло влезть. Можно было воспользоваться вкладками, а можно было нажать на специальную кнопку, которая увеличивала размер окна и открывала доселе скрытые элементы управления. Microsoft эти кнопки разонравились и, начиная с Windows 95, она планомерно удалила их из всех своих продуктов, заменив вкладками. Это и породило проблему. Раньше разные вкладки содержали примерно одинаково важные элементы, просто не все помещались в одно окно, а кнопка с треугольником скрывала элементы, про которые начинающий пользователь твердо знал, что они ему не нужны или пользоваться ими опасно. Теперь все во вкладках, поэтому пользователи часто уверены, что сразу невидное опасно.

В результате некоторые пользователи избегают пользоваться элементами, расположенными на изначально закрытых вкладках, даже если это ничем им не грозит. Поэтому нежелательно размещать на закрытых вкладках элементы, которые пользователям обязательно понадобятся, даже если эти элементы и не нужны постоянно (в этом случае правило про релевантность должно отступать). Разумеется, это не касается опытных пользователей.

В интернете и в остальных операционных системах, которым Microsoft была не указ, кнопки, увеличивающие размер окна и открывающие продвинутые элементы управления, сохранились в полном объеме. Учитывая тот факт, что никаких пользовательских проблем с ними не замечено, можно смело рекомендовать их и для использования в Windows, тем более что они позволяют добиться определенного брэндинга.

Похоже, что Microsoft сама осознала вред от потери увеличивающихся окон, вызванный недоверием пользователей ко вкладкам.

Число вкладок. Теоретически число вкладок может быть сколь угодно большим. На практике оно ограничивается двумя факторами: во-первых, объемом кратковременной памяти, а во-вторых, размером области, в которые ярлыки вкладок могут помещаться. Дело в том, что если ширина всех ярлыков будет больше ширины окна, придется либо делать несколько строк ярлыков, либо скрывать часть из них, пока пользователь не нажмет специальную кнопку. И то и другое плохо.

Несколько строк ярлыков плохо по двум причинам. Во-первых, из-за большого количества мелких деталей (границ ярлыков), вся конструкция довольно медленно сканируется, т.е. трудно найти нужную вкладку. Во-вторых, при последовательном обращении к нескольким вкладкам из разных рядов эти ряды меняются местами, т.е. каждый раз нужно заново искать нужную вкладку. И то и другое крайне негативно сказываются на субъективном удовлетворении и скорости работы.

Скрывать часть ярлыков тоже нехорошо. Предположим, что пользователь нажал на стрелку вправо, показывающую следующую часть ярлыков. Если при этом значительно пролистывать строку с ярлыками, пользователи будут полностью потерять контекст (сильнее даже, чем они теряют его, нажимая Page Down). Если же пролистывать строку по одному элементу, контекст не потеряется, но перемещение между вкладками будет очень медленным.

Существует и третий способ решения проблемы – можно просто убрать вкладки, заменив их раскрывающимся списком. Этот способ тоже не слишком хорош, поскольку не слишком визуален и к тому же сравнительно медлителен.

Похоже, что самым эффективным решением является комбинация второго и третьего способов: основные экраны реализуются в форме вкладок, а дополнительные вызываются через раскрывающийся список. Это позволяет обеспечить максимальное количество наглядности и скорости работы.

Объем содержимого. Фактически, каждая вкладка представляет собой отдельное диалоговое окно внутри другого диалогового окна. Поэтому странной выглядят попытки (встречающиеся огорчительно часто) рассортировать элементы управления так, чтобы во всех вкладках их было поровну. Делать это ни в коем случае нельзя. Один экран должен содержать только те элементы, которые в нем нужны и которые пользователь может в этом экране ожидать.

Не старайтесь рассортировать элементы так, чтобы в каждой вкладке их был одинаковое количество

Терминационные кнопки. В диалоговом окне с вкладками терминационные кнопки обязательно должны располагаться вне области вкладок.

Перемещение в пределах окна

Помимо навигации между экранами, существует еще и навигация внутри отдельных экранов. Пользователям необходимо дать возможность максимально быстро переходить к необходимым элементам управления. Для этого у них есть два способа – мышь и клавиатура. С мышью все более-менее понятно: закон оптимизация диалогового окна, уменьшающая дистанцию перемещения курсора, всегда приводит к росту (хотя и небольшому) производительности пользователей.

С клавиатурой сложнее. Пользователь может перемещаться между элементами управления двумя разными способами: клавишей Tab и горячими клавишами. Перемещаться клавишей Tab медленно, но зато для этого не нужно обращаться к памяти или высматривать клавиатурную комбинацию для нужного элемента. Напротив, горячие клавиши позволяют быстрее перемещаться вглубь экрана, но требуют запоминания клавиш. Таким образом, пользователи, которые часто вводят данные в какой-либо экран, стараются использовать клавишу Tab и только изредка пользуются горячими клавишами. Соответственно, любая форма ввода, которой часто пользуются, обязана корректно работать с Tab, при этом желательно, чтобы она работала и с горячими клавишами.

Работа пользователей с клавишей Tab может быть неприятна по двум причинам. Во-первых, на экране могут быть элементы, не подразумевающие взаимодействия с пользователем (например, скрытая или заблокированная кнопка, поле вывода), но на которые перемещение совершается. Избавиться от этой проблемы легко – нужно лишь явно указать, чтобы в список объектов, между которыми можно перемещаться, ОС их не вносила. Во-вторых, бывают ситуации, когда визуальный порядок элементов управления (происходящий из-за того, что пользователи читают экраны) не совпадает с порядком перемещения. В этом случае нужно просто сменить у неправильных элементов их место в последовательности.

Последовательные окна

Особым вариантом окон являются действия, выполняющиеся в последовательности сменяющих друг друга окон (мастера). Чтобы осознать правила, применимые к ним, полезно определить причины, вызвавшие появление таких окон.

Во-первых, существуют действия, для которых либо естественна, либо желательна жесткая последовательность. Для таких действий единый экран, в котором выполняется вся последовательность, не слишком эффективен: он грозит человеческими ошибками, к тому же, чтобы его использовать, требуется построить ментальную модель экрана (чтобы, как минимум, знать, что нужно сделать в начале, а что в конце). Эффективнее разбить действие на несколько разных экранов. Во-вторых, существуют действия, которые всегда будут вызывать проблемы у пользователей, либо потому, что эти действия сложны, либо потому, что нужны они редко (так что пользователям нет резона учиться). При этом единое окно для выполнения действия также оказывается неэффективным, поскольку объем справочной информации, которую в него нужно вместить, слишком невелик. В таких случаях разделение действия на последовательность экранов позволяет снизить насыщенность отдельных экранов и тем самым освободить место для справочной информации. Как правило, одной причины достаточно, чтобы оправдать использование мастера, когда же действуют обе причины, мастер становится обязательным. Итак, теперь, когда определены причины возникновения мастеров, можно перейти к конкретным правилам их создания.

Переход между экранами. Понятно, что пользователи должны получить возможность переходить не только на следующее окно в последовательности, но и на предыдущие окна. Менее очевидным является другое требование к мастерам: переход должен быть максимально легким. Задача раскладывается на две составляющие: во-первых, нужно реализовать возможность свободного перемещения по последовательности. Если экранов немного (3-5), то вполне можно ограничиться стандартными кнопками Назад и Далее. Если же экранов много, переход по этим кнопкам будет, как минимум, медленным. В таких случаях разумно дополнять кнопки раскрывающимся списком (при этом, возможно, исключая из него ещё не пройденные экраны), либо, если есть место, снабжать мастера списком всех экранов (отмечая текущий и пройденные экраны). Независимо от числа экранов в последовательности, необходимо информировать пользователей об объеме оставшегося действия (чтобы дать им возможность оценивать количество работы и тем самым повышать их чувство контроля над системой). Справедливости ради надо уточнить, что в длинных последовательностях показ объема оставшихся экранов может снизить мотивацию пользователей в начале действия, но повысить мотивацию в конце («осталось немного, не буду это бросать»).

Вторая составляющая – четкость перехода. Для пользователей мастер, т.е. последовательность экранов, кажется единым экраном, содержимое которого меняется. Эту иллюзию нужно поддерживать, поскольку она позволяет не сбивать контекст действий пользователя и поддерживать внимание на «сюжетно-важной» области экрана. Для этого размер и расположение окна мастера, а также расположение и облик всех повторяющихся элементов (таких как терминационные кнопки) нужно выдерживать неизменными на протяжении всей последовательности.

Контекст. В отличие от единого окна, в котором выполняется действие, в мастерах необходимо поддерживать контекст действий пользователя. Поскольку ранее сделанная работа скрыта, пользователи могут потерять контекст, что может замедлить действие (контекст придется восстанавливать). Степень потери контекста зависит от количества экранов, времени, которое пользователи проводят за отдельными экранами и от времени реакции системы. И если количество экранов в мастере редко превышает шести (а это небольшое число), то время, проведенное на пройденных экранах и, особенно, реакция системы (особенно в интернете), могут быть значительными.

Единственным же средством поддержания контекста является вывод текущего состояния данных в процессе выполнения мастера. Как правило, обычный текстовый список с предыдущими установленными параметрами работает плохо (к тому же редко вмещается в экран), визуализировать же изменения трудно, если вообще возможно. Таким

образом, лучше избегать длинных последовательностей (тем более что уровень мотивации пользователей при увеличении продолжительности действия снижается).

Вывод справочной информации. Благодаря обилию пустого места мастера замечательно подходят к выводу справочной информации в самом интерфейсе. Справочную же информацию нужно выводить двух типов, а именно краткое и более развернутое описания текущего шага. С развернутым описанием все просто. Где-нибудь снизу экрана (чтобы не сбивать фокус внимания пользователей) выводится один или два абзаца, рассказывающие стандартный набор: что, зачем и почему. С кратким же описанием сложнее. К сожалению, устоявшийся облик мастеров не имеет большого и заметного заголовка (этой проблемы, к счастью, нет в интернете, где вообще нет ничего устоявшегося).

Это неправильно. У каждого окна последовательности должен быть ясно видимый и бросающийся в глаза заголовок. При этом в отличие от обычных заголовков окна, он должен быть написан не описательно, но командно (сделайте то-то и то-то). Microsoft, в некоторых своих продуктах широко использующая мастера (называя это побуждающим пользовательским интерфейсом) вообще рекомендует считать заголовки важнейшими элементами мастеров. Особо подчеркивается, что заголовки экранов должны быть созданы и сформулированы до начала проектирования экранов, при этом содержимое экранов не должно выходить за рамки смысла заголовков. Поспорить с Microsoft в данном случае затруднительно.