НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЧЕЛОВЕКО-МАШИННОЕ ВЗАИМОДЕЙСТВИЕ»

Новосибирск 2006 г.

Содержание

1. Введение

1.1. Определение интерфейса

Интерфейс — это нечто большее, чем окна, пиктограммы, выпадающие меню и мышь. Необходимость проектирования интерфейса уже на ранних стадиях разработки продукта иногда упускается из виду. Другой фактор, который часто недооценивается, состоит в том, что все мы наделены познавательными аппаратами, имеющими между собой много общего. При разработке интерфейсов следует сперва учесть общие факторы, а потом уже рассматривать индивидуальные различия. Но, к сожалению, существующие на сегодня средства конструирования интерфейсов не позволяют подойти к задаче именно таким образом.

Далее выражения *интерфейс «человек-машина»* или *интерфейс «человек-компьютер»* будут сокращаться до *пользовательского интерфейса* или просто *интерфейса*. Многие считают, что термин *пользовательский интерфейс* относится только к современным графическим пользовательским интерфейсам (graphical user interface, GUI), основанным на окнах и меню, управляемых с помощью мыши. На самом деле способ, которым вы выполняете какую-либо задачу с помощью какого-либо продукта, а именно совершаемые вами действия и то, что вы получаете в ответ, и является интерфейсом.

1.2. Разработка интерфейса как часть общего цикла разработки

Применяемые сегодня методы разработки проектов зачастую не считаются с необходимостью разработки интерфейса. Это упущение может быть следствием того, что специалисты по разработке интерфейсов привлекаются к проекту слишком поздно, когда возможности улучшения качества взаимодействия между пользователем и продуктом большей частью уже потеряны. Интерфейсом удобнее всего заниматься именно на начальных

стадиях разработки. И если специалисты по интерфейсам привлекаются уже после того, как программное обеспечение спроектировано и определены его инструменты или когда разработка программы уже почти завершена, то их рекомендации могут потребовать переделки всей выполненной работы, что, естественно, является неприемлемым. Когда бюджет проекта уже исчерпан и рабочий план почти завершен, перспектива отказа от большей части или даже всего дизайна и готового кода, конечно, не может вызвать энтузиазма у менеджеров проекта.

Определив задачу, для которой продукт предназначен, сначала спроектируйте интерфейс, после чего приступайте к его реализации. Это повторяющийся процесс. Определение задачи будет меняться во время разработки интерфейса. Поэтому весь процесс разработки продукта будет проходить в соответствии с изменениями в задаче продукта и его интерфейсе. Здесь необходимо стремиться к максимальной гибкости. На первом этапе разработки следует определить, что именно должен сделать пользователь для получения того или иного результата и как система должна отвечать на каждое его действие.

Пользователи не задумываются над тем, как устроена машина, пока она справляется со своими задачами. При этом не имеет значения, какой именно процессор используется и является ли язык программирования объектно-ориентированным, многопоточным или, быть может, называется какими-то другими умными словами. Для пользователей важнее всего удобство и результаты. Но все, что они видят, — это интерфейс. Другими словами, *с точки зрения потребителя именно интерфейс является конечным продуктом*.

1.3. Определение человекоориентированного интерфейса

Интерфейс является ориентированным на человека, если он отвечает нуждам человека и учитывает его слабости. Чтобы создать такой интерфейс, необходимо иметь представление о том, как действуют люди и

машины. Кроме того, следует развить в себе способность чувствовать те трудности, с которыми сталкиваются люди. И это не всегда просто. Мы настолько привыкли к тому, как работают программы, что соглашаемся принять их методы работы как данность, — даже в тех случаях, когда их интерфейсы неоправданно сложны, запутанны, неэкономны и побуждают людей к ошибкам.

Многие из нас испытывают раздражение, например, от того, что для запуска (иначе говоря, загрузки) компьютера требуется какое-то время. При этом каждый согласился бы с тем, что сокращение или устранение задержки при запуске компьютера улучшило бы эффективность его использования, тем более что, пожалуй, не существует пользователя, у которого такая задержка не вызывала бы раздражение. Однако никогда не существовало технической необходимости в том, чтобы компьютер после включения начинал работать более чем через несколько секунд. Наши компьютеры долго загружаются только лишь потому, что многие дизайнеры и разработчики не потрудились сделать интерфейс в этом отношении ориентированным на человека. Кроме того, некоторые люди думают, что если компьютеры с медленной загрузкой продаются миллионами, то это якобы свидетельствует об их высокой производительности.

Принцип разработки, согласно которому программные продукты не должны вынуждать пользователя ждать без необходимости, можно считать очевидным и ориентированным на человека. Таким же является и стремление не подгонять пользователя. В общем виде этот принцип можно было бы сформулировать следующим образом: *«Ритм взаимодействия должен устанавливаться самим пользователем»*.

Не требуется обладать большими техническими знаниями, чтобы понять, что большая пропускная способность коммуникационных линий может ускорить передачу веб-страниц. Однако другие взаимосвязи иногда бывают не столь очевидны. Поэтому для разработчиков интерфейсов «человек-машина» важно знать внутренние механизмы технологии. В

противном случае у них не будет возможности оценивать достоверность утверждений, высказанных, например, программистами или специалистами по аппаратной разработке относительно осуществимости тех или иных элементов интерфейса.

2. Элементы когнитивной психологии

2.1. Когнетика

Руководства по разработке продуктов, взаимодействующих с нами физически, обычно содержат конкретную информацию, основанную на свойствах и возможностях человеческого скелета и органов чувств. Совокупность сведений в этой области составляет науку **эргономику**. На основе этих знаний можно проектировать стулья, столы, клавиатуры или дисплеи, которые с высокой степенью вероятности будут удобны для своих пользователей. Тем не менее, нельзя пренебрегать тщательным тестированием разрабатываемых продуктов. Вы не станете проектировать машину, обслуживание которой предусматривает, чтобы один человек оперировал двумя переключателями, расположенными в трех метрах друг от друга. Очевидно, что людей с такими физическими размерами не бывает. Тема эргономики в компьютерной технике, выходящая за пределы данного изложения, рассматривается в обзоре разработок интерфейсов, представленном в книге Мэйхью. В эргономике учитывается статистическая волатильность параметров человеческого тела. Можно спроектировать автомобильное кресло, подходящее для 95% населения, тогда как остальным 5% потенциальных покупателей автомобиля такие кресла покажутся неудобными. Создание кресла, которое можно было бы регулировать в широком диапазоне, в том числе для редких пользователей с ростом 1 м или еще более редких с ростом 2,5 м, было бы механически невыполнимым или же потребовало значительных затрат.

Большая часть машин, созданных нашей цивилизацией, были механическими и взаимодействовали с нами главным образом физически. Соответственно, наши физические ограничения сравнительно легко учесть. Постепенно человеческие изобретения стали иметь все большее отношение к области интеллектуальных задач, нежели физических. *Мы должны овладеть*

эргономикой сознания, если мы хотим создавать интерфейсы, которые могли бы хорошо работать. Удивительно, но мы часто не замечаем собственные ментальные ограничения, поэтому для определения границ возможностей нашего сознания мы должны прибегнуть к тщательному наблюдению и экспериментированию.

Изучение прикладной сферы наших ментальных способностей называется когнитивным проектированием, или **когнетикой**. Некоторые когнитивные ограничения очевидны: например, нельзя ожидать от обычного пользователя способности перемножать в уме 30-значные числа за 5 секунд, поэтому нет смысла разрабатывать интерфейс, который требовал бы от пользователя такой способности. Однако мы часто не учитываем другие ментальные ограничения, которые оказывают неблагоприятное влияние на нашу продуктивность при работе с интерфейсами «человек-машина», хотя эти ограничения присущи каждому человеку. Интересно отметить, что все известные компьютерные интерфейсы, а также многие некомпьютерные интерфейсы «человек-машина» разработаны с расчетом на некие когнитивные способности, которыми, как показывают эксперименты, мы на самом деле не обладаем. Большая часть трудностей, связанных с использованием компьютеров и подобных устройств, возникает скорее из-за низкого качества интерфейса, чем из-за сложности самой задачи или же недостатка старания или умственных способностей у пользователя.

Когнетика, так же как и эргономика, учитывает статистическую природу различий между людьми. Тем не менее, следует прежде всего рассмотреть сами ограничения, присущие нашим когнитивным способностям, поскольку знания об этих ограничениях пока мало находили практическое применение.

2.2. Когнитивное сознательное и когнитивное бессознательное

Использование таких терминов, как сознательное и бессознательное, которые имеют вполне определенное значение в психологии, философии и

истории и применяются для описания аспектов функционирования нашего мышления, может вызывать некоторые затруднения. В контексте технического проектирования имеет смысл пользоваться более ограниченными понятиями когнитивного сознательного и когнитивного бессознательного. Еще более точными были бы термины эмпирическое сознательное и эмпирическое бессознательное, однако более благозвучные варианты, предложенные Килстромом, уже стали общепринятыми. Понимание того, что мы обладаем этими двумя отдельными наборами ограниченных ментальных способностей, а также того, как они работают во взаимодействии с интерфейсами «человек-машина», важно для разработки интерфейсов в той же степени, в какой знание о размере и силе человеческой руки важно для разработки клавиатуры.

Дадим краткое определение: бессознательными называются те ментальные процессы, которые вы не осознаете в тот момент, когда они происходят. Когнитивное бессознательное — это не грандиозное мифическое порождение фрейдистской психологии, а явление, которое можно продемонстрировать с помощью простого эксперимента, о котором мы скажем далее. Несмотря на обилие книг, посвященных вопросам и парадоксам сознания, изложенный в этом разделе подход, взятый из книги Бернарда Баарса «Когнитивная теория сознания», позволяет избежать дилемм, свойственных данному вопросу, и обращает нас только к тому, что мы можем непосредственно наблюдать и о чем можем делать конкретные выводы. Как сказано в предисловии к этой книге, «в науке применяется одна испытанная временем стратегия: оставить на время философские вопросы и сосредоточиться на тех, которые являются эмпирически разрешимыми». Когнетика — это практическая дисциплина. Хотя теоретические исследования могут разъяснять неясное и, в конце концов, приводить к надежным практическим результатам, мы все же стремимся избегать их до тех пор, пока не почувствуем эти результаты. (Аналогичным образом изучение роста человеческих костей, которое может дать полезные сведения

с точки зрения эргономики, все же скорее относится к области физиологии, чем собственно к эргономике.)

Поскольку рассуждения о том, что является сознательным или бессознательным, кажутся весьма далекими от наших повседневных забот, попытаемся наглядно продемонстрировать их значение в обычной жизни. Попробуйте ответить на следующий вопрос: какая последняя буква в вашем имени? До тех пор пока вы не прочитали предыдущее предложение, вы, вероятно, не думали об этой букве и ее связи с вашим именем. Вы знаете (и уже давно знали), что это за буква и какое место она занимает в вашем имени, но вы не обращали на это своего внимания. Вы не думали, не размышляли об этом. Или, если пользоваться нашей терминологией, вы не осознавали это. Данная информация не запрашивалась, однако вы смогли ее получить, когда в этом возникла необходимость. То место, откуда была извлечена буква, мы будем называть когнитивным бессознательным. Когнитивное бессознательное — это необязательно какое-то физическое место, хотя оно должно быть представлено определенными физическими процессами в мозге. Представление о том, что бессознательное и сознательное должны соответствовать определенным областям мозга, не совсем верно. Скорее, они могут им соответствовать. Изменение состояния в момент осознания буквы — это один из возможных механизмов процесса. Другой возможный вариант — это наличие в мозге определенного указывающего устройства, и перемещение воспоминания или мысли из одной области мозга в другую вызывает также и перемещение этого указателя. Вполне возможно, что мысли и воспоминания распределены по всему мозгу, наподобие того, что происходит при голографической записи изображений.

В табл. 1 обобщены различия между когнитивным сознательным и когнитивным бессознательным. Из нее видно, что когнитивное сознательное включается в тех случаях, когда вы сталкиваетесь с ситуацией, которая кажется новой или представляет угрозу, или когда вам требуется принять

нешаблонное решение, т.е. такое, которое основано на происходящем именно здесь и сейчас. Понять логическое содержание проблемы возможно только в том случае, если вы осознаете наличие этой проблемы. Когнитивное сознательное работает последовательно и может оперировать только одним вопросом или контролировать только одно действие в течение некоторого промежутка времени. Человек может осознавать одновременно от 4 до 8 отдельных мыслей или объектов. Как правило, каждые несколько секунд сознательная память очищается.

Сознательное проявляется при решении ветвящихся задач. Необходимо сказать, что иногда трудно отличить ветвящуюся задачу от неветвящейся. Например, торможение по сигналу светофора может относиться и к тому и к другому типу задачи. С одной стороны, если вы просто реагируете на красный свет нажатием педали тормоза, данная задача является неветвящейся и, следовательно, обрабатывается когнитивным бессознательным. С другой стороны, если в момент, когда вы приближаетесь к светофору, на нем зажигается желтый сигнал, и поэтому вам требуется принять решение, пересекать ли перекресток без остановки или остановиться, то здесь уже вступает в действие когнитивное сознательное. Пока вы изучаете некоторую задачу, вы можете воспринимать ее как ветвящееся событие, требующее сознательного внимания. По мере повторения задачи ее выполнение может стать неветвящимся и автоматическим.

Таблица 1. Свойства когнитивного сознательного и когнитивного бессознательного

Свойство	Сознательное	Бессознательное
Инициируется	Чем-то новым	Повторением
	Нестандартными ситуациями	Ожидаемыми событиями
	Опасностью	Безопасностью
Используется	В новых обстоятельствах	В привычных ситуациях
Решает задачи	Принятие решений	Работа с неветвящимися задачами
Принимает	Логические утверждения	Логические или противоречивые

		утверждения
Функционирует	Последовательно	Одновременно
Управляется	Волей	Привычными действиями
Производительность	Небольшая	Огромная
Период функционирования	Десятки секунд	Десятилетия (всю жизнь)

2.3. Сознание и модели человеческого разума

Хотя трактовка когнитивных свойств сознания, предложенная Баарсом, является полезной, его теория разума, в которой моделями служат современные цифровые компьютерные структуры, имеет уже меньшую ценность. Такой подход сомнителен особенно по той причине, что мыслители издавна использовали последние достижения в области технологии в качестве моделей понимания человеческого существа, а со следующим шагом в технологическом развитии отказывались от них или же ограничивали их применение.

Дон Норман, когнитивный психолог, хорошо знакомый с компьютерами, говорил о человеческом разуме в терминах вычислительных устройств. Дж. Р. Андерсон еще в 1976 году построил модель ментальных операций на основе так называемых продукций — инструмента, который широко использовался в свое время для описания синтаксиса компьютерных языков. Хотя такие аналогии и могут быть полезны, некоторые ученые склонны придавать чрезмерное значение этой метафоре. В ближайшие несколько лет может появится большое количество психологических теорий, использующих архитектуру «клиент-сервер», Интернет и Всемирную сеть с ее гипертекстовыми ссылками и броузерами в качестве моделей работы отдельных аспектов сознания.

Следует с осторожностью относиться к этой тенденции, в противном случае мы можем создать аналогии, которые не будут иметь никакой нейрофизиологической основы. В XVII столетии Вселенная и ее обитатели часто описывались в терминах часового механизма. Однако на фоне нашего сегодняшнего понимания как часовых механизмов, так и организмов, эта

метафора утратила свою яркость. В XIX столетии метафора парового двигателя встречалась во многих философских трактатах о физиологии человека. Сегодня мы знаем, что ценность этой аналогии ограничивается главным образом объяснением метаболизма, поскольку метаболическую систему можно назвать тепловым двигателем организма.

Тема сознания рассматривается также в двух других известных книгах: «Новый разум императора» Роджера Пенроуза (Penrose, *The Emperor's New Mind*, 1989) и «Объясненное сознание» Даниела Денетта (Dennett, *Consciousness Explained*, 1991). Несмотря на то что чтение этих произведений довольно захватывающе, они, к сожалению, не представляют ценности с точки зрения разработки интерфейсов «человек-машина». Обобщая, можно сказать, что без непосредственного подтверждения того или иного примера функционального параллелизма следует избегать слишком буквального понимания метафор, в которых человеческий мозг уподобляется компьютеру.

2.4. Локус внимания

Вы можете до некоторой степени контролировать превращение бессознательных мыслей в сознательные, в чем вы убедились, переместив знание последней буквы вашего имени в сознательную область. Однако вы не можете намеренно перевести сознательные мысли в бессознательную область. «Не думай о белом слоне», — шепчет девочка, зная, что мальчик не сможет не думать об этом слоне. Но через некоторое время, если разговор не останавливается на слонах, мысли мальчика об этом животном перейдут в бессознательное. Когда это произойдет, он больше не будет обращать внимание на мысль о слоне — слон перестанет быть локусом внимания.

Здесь используется термин *локус*, поскольку он обозначает некоторое *место* или *область*. Термин *фокус*, который иногда используется в этом контексте, может вызвать неправильное представление о том, как работает внимание, потому что может быть понят как действие. Когда вы находитесь в бодрствующем и сознательном состоянии, вашим локусом внимания

является какая-то деталь или объект окружающего мира или идея, о которой вы целенаправленно и активно думаете. Различие между фокусом и локусом внимания можно понять на примере следующего предложения: «Мы можем целенаправленно сфокусировать наше внимание на каком-либо локусе». Тогда как *фокусировать* означает волевое действие, мы, тем не менее, не можем полностью управлять содержанием локуса нашего внимания. Если вы слышите, как позади вас внезапно взорвалась петарда, ваше внимание будет направлено на источник звука. Слово *фокус* также используется при обозначении объекта, который в данный момент выбран на экране. Ваше внимание может быть — или не быть — направлено на такого рода фокус, когда вы пользуетесь тем или иным интерфейсом. Из всех объектов или явлений окружающего мира, которые вы воспринимаете с помощью своих чувств или воображения, в каждый момент времени вы можете сконцентрироваться только на одном. Чем бы ни был этот объект, деталь, воспоминание, мысль или понятие, он становится локусом вашего внимания. В данном случае имеется в виду не только то внимание, которое можно активно обращать на что-либо, но также и пассивное восприятие потока поступающей информации или простое — переживание происходящего.

2.5. Одновременное выполнение задач

На языке когнитивной психологии любая задача, которую вы научились выполнять без участия сознания, становится *автоматичной*. Автоматизм позволяет выполнять сразу несколько действий одновременно. Все одновременно выполняемые задачи, за исключением не более чем одной, являются автоматичными. Та задача, которая не является автоматичной, естественно, находится непосредственно в локусе вашего внимания. Когда вы выполняете одновременно две задачи, ни одна из которых не является автоматичной, эффективность выполнения каждой из них снижается в результате конкуренции за область внимания. Этот феномен психологи называют *интерференцией*. Чем более предсказуемой, автоматичной и

бессознательной становится задача, тем больше становится эффективность ее выполнения одновременно с другими задачами, и, тем менее, она конкурирует с ними.

Человек, по-видимому, имитирует одновременное выполнение нескольких задач, требующих сознательного контроля, через последовательное переключение внимания с одной задачи на другую. Действительная одновременность достигается, когда все задачи, кроме разве что одной, становятся автоматичными. Например, вы можете одновременно не спеша идти, что-нибудь есть и при этом решать какую-нибудь математическую задачу. (В это же время можно бессознательно обдумывать и еще одну математическую задачу, но по определению когнитивного бессознательного вы не заметите этого процесса. Здесь следует обратить внимание только на то, что вы не можете *сознательно* работать над двумя разными математическими задачами одновременно.) Для большинства людей все эти действия, за исключением поиска решения математической задачи, настолько знакомы, что могут выполняться «на автопилоте». Однако если при одновременном выполнении всех этих действий вы внезапно почувствуете какой-нибудь неприятный на вкус кусочек вашей походной еды, вы станете думать только о том, что вы такое съели, тогда как математическая задача перестанет быть вами осознаваемой.

Не менее важным, чем понимание того, что в каждый отдельный момент времени нельзя осознавать более одной задачи, является тот факт, что человек не может избежать формирования автоматических реакций. Эта невозможность не зависит от повторения: никаким количеством повторений нельзя научиться *не* формировать привычки при регулярном использовании того или иного интерфейса. Формирование привычек является неотъемлемой частью нашего ментального аппарата. Его невозможно остановить волевым действием. Наверное, когда-нибудь в субботу утром вы нечаянно приезжали туда, где находится ваша работа, хотя собирались поехать в какое-то другое место. Сделали вы это по привычке, которая сформировалась через

повторение определенной последовательности действий. Когда вы учились читать, то поначалу проговаривали по отдельности каждый слог и обращали внимание на произношение каждой буквы. Теперь же вы можете читать без необходимости сознательного контроля над процессом составления слов из букв.

Любая последовательность действий, которую вы регулярно выполняете, становится, в конце концов, автоматичной. Набор действий, составляющих последовательность, становится как бы одним действием. Как только вы начнете выполнять некоторую последовательность, требующую не более 1 или 2 секунд времени, вы не сможете остановиться и проделаете все действия вплоть до завершения последовательности. Вы также не сможете прервать последовательность, выполнение которой занимает больше нескольких секунд, если она не стала локусом внимания. Если возвратиться к примеру с субботним утром, то после того как вы сделали неверный поворот, вы могли внезапно осознать, что собирались ехать совершенно в другом направлении. Это осознание помещает текущую задачу управления автомобилем в локус вашего внимания и позволяет вам прервать автоматичную последовательность действий, которая направила вас к месту вашей работы.

Когда вы повторяете какую-то последовательность действий, единственный способ предотвратить формирование привычки — это *удерживать* то, что вы делаете, в локусе внимания. Это очень сложно. Как обычно говорят, наше внимание «гуляет».

3. Принципы проектирования пользовательского интерфейса

Далее приводятся шесть принципов, вобравших в себя многое из того, что на данный момент известно о разработке эффективного пользовательского интерфейса. Каждый из них включает в себя несколько связанных между собой идей, более детализированных по сравнению с общими вопросами. Этими общими вопросами являются структура, простота, видимость, обратная связь, толерантность и повторное использование.

3.1. Структурный принцип

Организаций пользовательского интерфейса должна быть целесообразной, осмысленной и удобной. Она должна базироваться на четких, целостных моделях, очевидных и распознаваемых пользователями. При этом родственные понятии должны быть связаны, а независимые — разделены. Непохожие элементы должны дифференцироваться, а похожие — выглядеть похоже.

Структурный принцип связан с общей архитектурой интерфейса и напрямую отражает представление о пользовательском интерфейсе как о диалоге между разработчиками и пользователями. Организация хороших интерфейсов продумывается очень тщательно, таким образом, чтобы отражать структуру решаемых системой задач и способ мышления пользователей относительно этих задач. Очень часто, особенно при использовании современных визуальных сред разработки, расположение визуальных компонентов внутри форм или диалогов и их распределение между ними оказывается почти случайным и отражает в лучшем случае последовательность, в которой программистами затрагивались те или иные вопросы. По идее, свойства и функции, которые чаще всего используются совместно или рассматриваются пользователями как связанные друг с другом, должны располагаться вместе или, по крайней мере, должны быть

четко и ясно взаимосвязаны. Что же до тех элементов, которые в контексте задачи или в сознании пользователя никак не связаны между собой, то они должны быть разнесены в интерфейсе. Подобное должно быть подобно. Похожая информация должна быть организована с помощью похожих решений, а объекты, обладающие похожим поведением, должны иметь общее представление.

Памятуя о диалоговом принципе проектирования, следует создавать интерфейсы, предоставляющие пользователю четкую и цельную модель, понятную для него и узнаваемую им. Во многих случаях с целью проектирования действительно хорошей структуры берутся объекты и действия, знакомые пользователю или напоминающие ему о знакомых вещах. Это, впрочем, не может служить оправданием для создания глупых, упрощенных симуляторов материальных объектов реального мира. Очень часто они оказываются неэффективными или попросту негодными.

Метафоры, заимствованные из реального мира, такие как папки для хранения файлов, кабинеты и комнаты, в рамках которых организуются данные, могут быть полезными для структурирования пользовательских интерфейсов и повышения их доступности, особенно для начинаю них пользователей. Метафоры чрезвычайно популярны как среди специалистов по практичности, так и среди общественности, покупающей программную продукцию, однако они не являются универсальным решением всех вопросов проектирования интерфейсов. Натянутые, неестественные имитации могут сделать интерфейс непонятным, а упрощенные модели – усложнить пользование системой.

Следует понять, что важен вопрос не наличия метафоры или даже целого набора метафор, включенных в интерфейс, а правильной организации и соответствия выполняемым задачам. Если метафора напрашивается сама, как только становится ясна суть решаемой задачи, можно рассмотреть вопрос о ее применении, однако, право, не стоит расстраиваться из-за того, что не удается найти метафору. Если она не приходит на ум сразу, значит, либо ее

не существует, либо она не слишком полезна. Центральным звеном при создании структуры интерфейса все равно остается пользователь — то, как он мыслит, как работает, какие задачи решает.

3.2. Принцип простоты

Следует максимально упрощать управление паи бол ее распространенными операциями. При этом общение с пользователем должно вестись на понятном для него языке. Должны предоставляться ссылки, логичным образом указывающие на более сложные процедуры.

Говорить, что чем проще интерфейс, тем он лучше, — это все равно, что говорить, что чем солнечней день, тем он светлее. На самом деле нам всем хотелось бы создавать и пользоваться исключительно простыми интерфейсами, однако, к сожалению, очень часто оказывается невозможным упрощение как интерфейсов, так и решаемых системой задач. Процесс проектирования интерфейса — это всегда борьба за компромисс Упрощение чего-то одного неизбежно приводит к усложнению чего-то другого. Если уменьшить количество меню, увеличится число пунктов в каждом из них. Если сделать маленькими все диалоговые окна, включив в них как можно меньше элементов, любое взаимодействие пользователя с системой обернется для него необходимостью обращаться к большому количеству таких окошек.

Невозможно сделать все на свете простым. Следование принципу простоты требует от вас знания того, какие задачи выполняются пользователем наиболее часто и какие из них, с точки зрения пользователя, проще. Именно такие задачи следует упрощать, чтобы пользователь мог быстро их решить.

3.3. Принцип видимости

Все функции и данные, необходимые для выполнения данной задачи, должны быть видны, чтобы пользователь не отвлекался на дополнительную и избыточную информацию.

Принцип видимости связан с проектированием таких пользовательских интерфейсов, в которых видны все элементы, нужные для выполнения данной задачи. Цель — перейти от философии WYSIWYG (What You See Is What You Get - Что видишь на экране, то и получишь в результате) к философии WYSIWYN (What You See Is What You Need — На экране видишь то, что тебе нужно). Интерфейсы WYSIWYN оставляют видимыми те и только те элементы, которые действительно нужны пользователю для выполнения операции.

С одной стороны, в задачи проектирования входит создание такого интерфейса, на котором были бы явно видны все нужные и иажные функции. С другой стороны, хороший интерфейс не должен заваливать пользователя слишком большим количеством возможных вариантов или смущать его избыточной информацией. WYSIWYN-интерфейсы лучше уже тем, что они принимают во внимание ограниченность объема «оперативной памяти» человека и способность узнавать вещи быстрее, чем вспоминать. Нагрузка на долговременную память уменьшается за счет того, что пользователь постоянно видит все необходимые опции и варианты. На кратковременную память нагрузка снижается за счет того, что пользователю не приходится запоминать и затем воспроизводить информацию, содержащуюся в какой-то другой части интерфейса.

Уж сколько раз нам приходилось ждать у конторки портье в гостиницах, наблюдая, как администратор списывает информацию с экрана на клочок бумаги, чтобы затем снова ввести ее в компьютер, переключившись на новое окно. Можно больше ничего не знать о приложении — уже понятно, что это полный провал программной разработки. Опять же, программисты просто не удосужились задуматься над тем, как их система будет применяться и какая информация будет нужна на разных этапах. Они, скорее всего, мыслили в терминах базы данных, ломали голову над организацией колонок и строк, долго подбирали толщину сетки в таблицах, однако даже не вспомнили о

несчастном администраторе, пытающемся найти свободный номер для некурящих.

3.4. Принцип обратной связи

Сообщайте пользователям о действиях системы, ее реакциях, изменениях состояния или ситуации, об ошибках и исключениях, которые важны для них. Сообщения должны быть четкими, краткими, однозначными и написанными на языке, понятном пользователю.

Хорошие пользовательские интерфейсы находятся в диалоге с пользователями, сообщая им о том, что происходит в системе. Принцип обратной связи указывает разработчикам некоторые правила этого диалога.

Каждому из нас приходилось бывать в таких ситуациях, когда мы кому-нибудь о чем-нибудь говорили, а потом выяснялось, что мы не были услышаны. Сообщение, которое тяжело увидеть или услышать, принесет мало пользы. Составной частью успешной обратной связи с пользователем является представление информации в таком виде, что ее легко заметить, прочесть и корректно воспринять.

Где на экране должны располагаться элементы обратной связи? Везде, куда смотрит пользователь. А куда смотрит пользователь? Скорее всего, туда, где находится указатель мыши или курсор.

Помимо той области, в которой пользователь обычно работает, обратная связь ожидаема либо в центре экрана, либо сверху, либо где-то у нижней границы. Стандартное расположение информации об изменениях состояния соответствует строке состояния внизу экрана. Однако такая практика, несмотря на свою популярность, не является удачной, особенно если руководство по стилям предписывает оформлять эту строку обычным шрифтом на сером фоне. Многие из нас сталкивались с тем, что не замечали сообщения внизу экрана, а потом удивлялись, почему программа ведет себя как-то странно.

Если вы вынуждены в силу каких-то обстоятельств (договоренностей с заказчиком или упрямым начальником) размещать важную информацию в строке состояния, позаботьтесь о выгодном использовании других психологических и визуальных свойств человека. Пользователь должен заметить ваше послание к нему, а его внимание может привлечь движение и выделяющийся цвет. Если строка состояния при появлении несколько раз «моргнете ярким цветом, это повысит вероятность того, что она будет замечена. Конечно, не стоит впадать в крайности и делать строку постоянно моргающей то желтым, то синим цветом, однако одна-две «вспышки» помогут уменьшить количество ошибок пользователя.

Практичные системы информируют пользователя о множестве вещей. К примеру, они должны позволять ему узнавать о том, как воспринимаются вводимые им данные. Всякий раз, когда меняется внутреннее состояние системы, и это может оказать какое-либо влияние на работу пользователя, его следует уведомлять об этом, особенно если меняется интерпретация системой его действий. Разумеется, пользователь должен знать о действиях, которые запрещены или игнорируются. При этом принцип обратной связи не может служить оправданием созданию бесконечных окошек сообщений. Информирование пользователя — не самоцель, а способ организации диалога в компактной и естественной форме.

Пользователям также требуются сообщения об ошибках и исключительных ситуациях. Во многих программах эти сообщения, к сожалению, неинформативны и способны ввести в заблуждение. Можно иногда встретить даже оскорбляющие сообщения, после прочтения которых пользователю может стать не по себе. Вряд ли человек вдохновится, скажем, такой надписью: «Неправильно! Введите корректные данные!». Такое сообщение не только неявно предполагает, что пользователь — какой-то нехороший человек, но и, по сути дела, не дает никакой информации. Здесь не сказано, что именно неправильно и почему.

Грамотно составленные сообщения об ошибках — это еще один пример хорошей организации общения с пользователем. Рекомендации здесь можно дать такие: краткость; язык, понятный пользователю; простота понимания. Прежде всего информативным должен быть заголовок сообщения. Он должен в сжатой форме описывать проблему, а уже само сообщение должно раскрывать подробности и предлагать способы решения или последовательность корректирующих действий.

3.5. Принцип толерантности

Интерфейс должен быть гибким и толерантным. Ущерб, наносимый ошибками пользователи, необходимо снижать за счет возможности отмены и повтора действий и за счет предотвращения появлений этих ошибок путем анализа различных форматов ввода и разумной интерпретации любых разумных действий.

Уменьшение числа ошибок — дело еще более благородное, чем создание грамотных сообщений об ошибках. Чем практичнее система, тем сильнее она помогает пользователям избежать ошибок. Есть множество стратегий уменьшения ошибок пользователя, и они сведены воедино в принципе толерантности.

Благородство заключается в умении прощать и быть уступчивым. Это относится и к программному обеспечению и касается не только обработки различных форматов ввода и действий пользователя, но и подавления желания наказать пользователя, когда он делает что-то не то. Самыми неудачными являются системы, которые просто повисают или останавливаются в нерешительности при возникновении неожиданной ситуации, однако повсеместно встречаются и менее экстремальные примеры отсутствия толерантности в поведении программных продуктов. Ошибки необходимо предвидеть, к ним нужно быть готовым. Опытные программисты и разработчики знают, что пользователь — это человек, а человеку свой-

ственно ошибаться. Более того, они знают, что программы пишутся людьми, а значит, ошибки в них также неизбежны.

Не все, что программистами воспринимается как ошибка, является ошибкой на самом деле. Во многих случаях при программировании используется слишком жесткая логика, предъявляются необоснованно узкие ограничения на принимаемые от пользователя данные. Купер, говоря об уменьшении количества сообщений об ошибках, на самом деле говорит о толерантном, лояльном программировании. Хорошее программное обеспечение разумно реагирует на любые осмысленные действия пользователя. Невозможно, да и необязательно предусматривать абсолютно все варианты как вводимых данных, так и действий. Может быть, нет необходимости в том, чтобы программа всегда вела себя идеально и принимала наимудрейшие решения при возникновении любых неординарных ситуаций. Однако программа, по крайней мере, не должна делать глупости в ответ на неожиданные действия пользователя. При работе со многими задачами разные пользователи могут находить разные пути решения, и толерантное ПО не должно препятствовать этому.

Многие программные системы — даже стандартные операционные системы — регулярно отказываются воспринимать вполне разумные действия пользователей. Вот самый стандартный случай: вы проходите несколько уровней вложенных диалоговых окон и вдруг осознаете, что попали не туда Нажатие на, казалось бы, специально предназначенной для выполнения задуманного кнопке или пункте меню приводит к раздражающему гудку или звонку, напоминающему о том, что следует закрыть все ранее открытые вложенные окна, прежде чем пытаться сделать что-то еще. Для этого приходится истыкать весь монитор, нажимая на кнопки с надписями Отмена, ОК или Закрыть. Если подумать, то такая «ошибка», совершенная пользователем, в общем-то очень естественна и встречается на каждом шагу. В большинстве случаев система и сама могла бы озаботиться закрытием всех ненужных окон, и это было бы разумным ответом

пользователю, который говорит. «Мне сюда не нужно, мне здесь нечего делать; хочу вернуться обратно и сделать кое-что другое!»

Интерфейс можно делать более или менее толерантным в зависимости от того, какие данные проверяются и когда. Проверка всех полей разом по окончании ввода данных — практика распространенная и иногда оправданная. При этом толерантности системе добавит автоматическая подсветка поля с неправильными данными, установка на него курсора, плюс короткое, информативное сообщение в строке состояния. Больше всего пользователи страдают от программ, которые по окончании ввода во все поля проверяют всю форму, и в случае неправильных данных хотя бы в одном из полей пользователь оказывается снова один на один с пустым бланком. Казалось бы, такое решение ужасно нелепо, но оно встречается очень часто, в том числе и в коммерческих программах.

Проверка данных — это необходимая процедура, но нельзя доводить ее до абсурда, если вы не хотите нанести ущерб практичности. Например, в первой версии компьютерной системы приема в колледж все поля ввода проверялись программой. В результате приемная комиссия признала систему практически непригодной. При первичном вводе данных значения многих полей могли быть еще не известны, и служащим приходилось вводить какие-то фиктивные данные только для того, чтобы заполненная форма могла пройти проверку. Несмотря на то что многие поля не были обязательными и официально не использовались, для них также проводилась проверка, а уж это было и вовсе ни к чему, к тому же сильно замедляло процесс ввода данных.

В целом проверка неиспользуемых полей или полей, которые никак не обрабатываются системой и представляют интерес только для пользователей (в том виде, в каком они были введены), ведет к снижению толерантности ПО. Например, проверка того, что в поле примечаний присутствуют только буквенно-цифровые символы, избыточна Кроме того, если пользователь

вдруг захочет выделить что-нибудь в этом поле спецсимволом или с помощью псевдографики, у него возникнут проблемы.

3.6. Принцип повторного использования

Следует многократно использовать внутренние и внешние компоненты и принципы поведения системы, поддерживая устойчивость осмысленно, а не просто за счет избыточности. Это способствует уменьшению объема информации, которую пользователям приходится запоминать и о которой приходится думать каждый раз заново.

Об устойчивости ходит столько разговоров, что практически все воспринимают это как неотъемлемое свойство хорошего пользовательского интерфейса. Непротиворечивость расположения видимых элементов, целостность их внешнего вида и поведения внутри интерфейса — все это делает программное обеспечение более простым в изучении и в запоминании того, как им пользоваться. Примечательно, что принцип, о котором столько говорят, на практике реализуется в большинстве случаев весьма слабо. По правде говоря, достичь целостности внутри сложного пользовательского интерфейса действительно весьма непросто. Более того, интерфейсы, обладающие хорошей устойчивостью, зачастую достигают ее ценой ухудшения других важных критериев.

Однако со временем стало понятно, что именно повторное использование, а не целостность является ключевым вопросом. На самом деле с улучшением реализации идеологии повторного использования повышается и устойчивость. Повторное использование компонентов и приемов проектирования как внутри, так и между системами — это полезное свойство с точки зрения пользователей, так как им приходится изучать и запоминать меньшие объемы информации, а кроме того, такой интерфейс более предсказуем и понятен. Создание пользовательских интерфейсов и поддержка их внутренней структуры за счет повторного использования ком-

понентов гарантирует целостность не только внешнего представления, но и поведения системы. Знания, имеющиеся у пользователей относительно данной системы или других систем, собранных иэ подобных компонентов, скорее всего, будут применимы везде, где будут встречаться знакомые компоненты и решения.

Применяя повторное использование внешних и внутренних компонентов и решений, распространяющихся на всю систему, разработчик может создать не только более целостный интерфейс, но и более дешевый продукт. Стремление к одной лишь устойчивости повышает стоимость разработки, да и в ряде случаев оказывается сизифовым трудом. Нужно стремиться к устойчивости в контексте решаемых системой задач и области ее использования, устойчивость ни в коем случае не может быть самоцелью, иначе она может выглядеть несколько глуповато и даже приводить, как ни странно, к неудачным с точки зрения непротиворечивости системам.

Многие принятые стандарты и общепризнанные компоненты пользовательского интерфейса являют собой примеры неудавшихся попыток реализации устойчивости. Стандартные программные платформы порой навязывают разработчика плохо продуманные и неудачно спроектированные решения. Самые обычные диалоговые окна, скажем, могут обеспечивать целостность и при этом быть весьма низкосортными, выбор стандартных горячих клавиш оказывается случайным и никак не соответствующим принципу устойчивости.

3.7. Другие правила

Разумеется, одиннадцатью правилами и принципами, перечисленными ранее, не исчерпывается список указаний, помогающих проектировщикам в создании более практичного ПО. Якоб Нильсен, ярый сторонник простых и выгодных с экономической точки зрения подходов к практичности, создал один из лучших наборов эвристических правил практичности, а затем занялся изысканиями в области применения различных правил для пояснения

актуальных проблем практичности, возникающих в реально существующих системах. Общее число специальных эвристических правил равно 101. Ниже перечислены десять правил Нильсена, объясняющих 95% наиболее серьезных из этих проблем.

1. Зрительное восприятие/указание, а не запоминание/набор на клавиатуре.
2. Непротиворечивость (похожие действия реализуются похожим образом).
3. Своевременная и продуманная обратная связь.
4. Ярко выраженный набор действий.
5. Снисходительность (возможность отмены действий).
6. Близкая пользователю концептуальная модель.
7. Обратная связь (подтверждение введенных данных).
8. Предотвращение возможных ошибок.
9. Четко различимый выбор действий.
10. Немодальное взаимодействие.

В дополнение к этим десяти правилам существуют еще пять, поясняющие все проблемы независимо от степени их серьезности.

1. Общение с пользователем должно вестись на его языке.
2. Эстетическая целостность (простой дизайн).
3. Сокращенный клавишный набор и ускоренный доступ к командам.
4. Договоренности, имеющие практический смысл.
5. Помощь в обнаружении и исправлении ошибок.

Все эти правила, считающиеся лучшими из лучших, в том или ином виде можно узнать в пяти правилах и шести принципах практичного проектирования. Несколько нетривиальным требованием может показаться немодалъное взаимодействие, но и его можно свести к принципу толерантности, поскольку немодальные диалоги не накладывают жестких ограничений на последовательность действий пользователя.

Хотя факторный анализ правил и не выявляет каких-либо несомых факторов, первые семь факторов, покрывающие около 30% этой своеобразной дисперсии, несомненно, оказываются весьма знакомыми.

1. Наблюдаемость состояния системы (оповещение пользователя).
2. Согласованность системы с реальным миром (использование языка пользователя и договоренностей, имеющих практический смысл).
3. Пользовательский контроль над системой и свобода (легкий выход, отмена и повтор действий).
4. Непротиворечивость и следование стандартам.
5. Предотвращение ошибок.
6. Узнавание, а не вспоминание (уменьшение объема информации, которую
   необходимо запоминать, за счет применения визуальных элементов, определяющих свойства, действия и команды).
7. Гибкость и эффективность использования (настройка системы и поддержка опытных пользователей).

Нильсен, кроме того, добавил еще три правила, завершив тем самым пересмотр набора из десяти эвристик.

1. Эстетичный, минималистский дизайн (сокращение не относящейся к делу
   или редко используемой информации).
2. Помощь в распознавании, диагностике ошибок и восстановлении после них.
3. Хорошая справочная система и документация.

Нетрудно догадаться, что существует связь между всеми этими эвристиками, правилами и принципами практичности. Исключение составляет лишь последнее правила Справочные системы и документация оказывают огромное влияние на практичность, но этот аспект рассматривается в качестве одной из частей интерфейса между системой и

пользователем, в которой находят применение самые широкие понятия практичности.

Надо заметить, что разработчики, обучающиеся применению базовых правил и принципов практичного проектирования, находятся на верном пути. Эти одиннадцать простых эвристик построены на знаниях о человеко-машинном взаимодействии и подкрепляются большим практическим и исследовательским опытом. Все вместе они охватывают действительно большой круг вопросов. Последовательное применение их при принятии решений в процессе проектирования позволяет надеяться на появление в будущем значительно более практичных программных систем.

4. Моделирование задач с использованием *use case* диаграмм

4.1. Моделирование задач

Есть много способов моделирования действий. Самыми традиционными способами описания задач являются различные модели, которые должны быть знакомы разработчикам программного обеспечения. Одним из популярных подходов является создание своеобразных блок-схем, описывающих процессы в виде логических последовательностей событий или процессов. Даже большие, сложные задачи можно описать, разбив их на ряд мелких, последовательных шагов. Эта идея лежит в основе всех компьютерных программ, всего программирования вообще. Блок-схемы, конечные автоматы, модели технологических процессов и диаграммы потоков данных — все это лишь различные формы последовательных моделей, которые могут использоваться и используются для представления задач, решаемых людьми.

4.2. От сценариев к моделям use case

Сценарий — это еще один способ описания структуры задачи. Это повествовательный рассказ о совершаемых действиях, это история, эпизод, происходящий в данных временных рамках и в данном контексте. Различные формы сценариев широко применяются при разработке программного обеспечения. Сценарии задач и взаимодействий обычно богаты характеристиками и обладают высокой реалистичностью.

Сценарии при разработке пользовательского интерфейса описывают взаимодействие между пользователем (или типом пользователей) и системой. Опыт подсказывает, что обыкновенные сценарии обладают некоторыми серьезными ограничениями при попытке использовать их для проектирования пользовательского интерфейса. В них делается упор на реалистичность и детали, при этом на серьезные проблемы и общую

организацию обращается недостаточно внимания. Сценарии включают в себя правдоподобные описания комбинаций отдельных действий и задач, поэтому часто бывает тяжело выделить и понять основную суть взаимодействия.

Концепция моделей use case впервые была применена для разработки ПО Айваром Якобсоном в качестве составной части его объектно-ориентированного подхода к программной инженерии. Успех модели оказался столь значительным, что со временем произошла интеграция элементов use case практически во все основные методы объектно-ориентированного анализа и проектирования. Несмотря на то что модель была разработана специально для проектирования объектно-ориентированного ПО, ничего особенно «объектно-ориентированного» в элементах use case нет, поэтому их можно применять практически ко всем подходам к проектированию.

Элемент use case — это ситуация, вариант использования, то есть некоторый случай применения системы. Что такое use case? Это:

– обеспечение функциональности;
– сугубо внешняя точка зрения (принцип «черного ящика»);
– повествовательное описание;
– описание взаимодействия между пользователем (в какой-то роли) и системой;
– завершенное и понятное пользователю применение системы.

Будучи термином, используемым в объектно-ориентированной программной инженерии, элемент use case является повествовательным описанием взаимодействия между пользователем (в какой-то роли) и некоторой системой. Элементы use case служат «черными ящиками», то есть рассматривается лишь обеспечиваемая ими функциональность, предоставляемая системой пользователю. Их можно рассматривать как часть набора требований или спецификации. Помимо их применения для разработки набора требований, элементы use case играют центральную

организующую роль во всем процессе разработки при использовании некоторых объектно-ориентированных подходов.

Каждый элемент use case описывает в повествовательной форме завершенное, хорошо определенное взаимодействие, имеющее ясную цель с точки зрения пользователя. При объектно-ориентированном подходе элементы use case могут описывать взаимодействие с другими системами и оборудованием, а не только с живыми пользователями. Тем не менее, когда целью является разработка пользовательского интерфейса, мы можем совершенно спокойно ограничиться рассмотрением только тех элементов use case, которые относятся к взаимоотношениям человека и системы.

Элементы use case, как изначально и задумывалось! часто выражаются в виде непрерывного, линейного описания. Нам кажется, что соглашение, введенное специалистом по методологии Ребеккой Уирфс-Брок (Rebecca Wirfs-Brock), имеет множество преимуществ. В предложенном ею структурном виде описание делится на две части: модель действий пользователя (показывающую, что делает пользовать) и модель реакций системы (показывающую, как система реагирует на дебсгвия пользователя). В таком виде наблюдается четкое разделение обязательств и интересов пользователя, системы и ее разработчиков.

Рассмотрим процесс получения наличных денег в банкомате. Эта задача часто приводится в качестве примера моделирования с помощью элементов use case. Вот как она решается банкоматами австралийских банков (табл. 4.1).

Таблица 4.1. Прецедент получениеДенег

Действие пользователя	**Реакция системы**
Вставка карточки	
	Считыванеи данных с магнитной полосы
	Запрос PIN
Ввод PIN	
	Проверка PIN
	Вывод меню возможных транзакций
Нажатие клавиши	
	Вывод меню расчетов
Нажатие клавиши	
	Запрос суммы

Ввод суммы	
	Вывод суммы
Нажатие клавиши	
	Возврат карточки
Получение карточки	
	Выдача денег
Получение денег	

4.3. Сущностные элементы use case

Некоторые разработчики и специалисты по методологии предпринимали попытки использовать элементы use case в их исходном виде, в качестве вспомогательного средства при разработке пользовательского интерфейса и делали это с большим энтузиазмом и с переменным успехом. Наши ранние работы и эксперименты выявили ряд больших проблем и ограничений, связанных с использованием элементов use case для этих целей. В частности, традиционно элементы use case содержат слишком много предположений, зачастую скрытых или неявных, относительно того, каким должен быть пользовательский интерфейс. Как модели они слишком сильно опираются на реализацию и в недостаточной мере заботятся о проблемах, выявленных пользователями.

Например, только что показанный элемент use case для задачи получениеДенег предполагает, что используются кредитные карточки с магнитной полосой, видеоэкран и некая клавиатура, с помощью которой пользователь вводит данные. В нем есть шаги, которые пользователю не нужны, но которые, тем не менее, появляются благодаря соответствующему интерфейсу и внутреннему устройству программы. То есть получается, что и интерфейс, и внутреннее устройство также определяются заранее. К сожалению, эти предположения в явном виде вы в модели не встретите. В этой традиционной форме элементов use case некоторые аспекты интерфейса определены заранее и уже как бы не требуют проектирования. Такие неявные решения вместе с ограничениями, которые они налагают на конечный вид пользовательского интерфейса, могут оказаться далекими от идеала при

решении конкретной задачи получения денег в банкомате. Если мы будем строить пользовательский интерфейс, базируясь на таких конкретных и специфических описаниях, у нас мало шансов создать действительно хорошую систему.

ОПРЕДЕЛЕНИЕ

Сущностный элемент use case — это структурированное повествование, выраженноа на языке данной прикладной области и пользователей системы и содержащее упрощенное, обобщенное, абстрактное, не зависящее от технологии и реализации описание одной завершенной, наполненной смыслом и хорошо определенной с точки зрения пользователей задачи или взаимодействии. Предполагается, что пользователь играет определенную роль по отношению к системе, а в описании воплощаются цели и замыслы лежащего в его основе взаимодействия.

Сущностные элементы use case строятся на основе целей и задач пользователя, а не на основе каких-то конкретных механизмов или этапов, ведущих к достижению этих целей. Некоторым кажется значимым включение целей пользователей в модели use case, но это не должно быть связано с упрощениями, присущими сущностному моделированию.

При использовании подхода, ориентированного на удобство использования, в сущностных элементах use case, являющихся структурированным описанием, можно выделить три части: изложение общих устремлений пользователя, выраженное в элементе use case, плюс состоящее из двух частей описание, включающее в себя модель пользовательских устремлений и модель обязательств системы. Сущностные элементы use case именуются, причем при помощи этих имен стараются выразить пользовательские намерения в условиях данного варианта использования.

Обозначаются сущностные элементы use case очень лросто. В соответствии с соглашением, предложенным Якобсоном, элемент use case

изображается в виде эллипса с именем элемента (см., например, рис 4.1 далее по тексту). В соответствии с соглашением, предложенным Уирфс-Брок, описание элемента use case организуется в виде двух параллельных столбцов. Если в обычных элементах use case эти столбцы назывались «моделью действий пользователя» и «моделью реакций системы», то в *сущностных* элементах use case названия этих столбцов другие. Они отражают изменения, произошедшие в связи со сменой используемой модели. Так, левый столбец сменил название на «модель пользовательских устремлений», поскольку в сущностной модели упор смещается с действий на устремления, то есть на «цели, направляющие действия, «образ действия, которому намерен следовать пользователь». Правая колонка становится «моделью обязательств системы», отражая в своем названии ожидания пользователя, касающиеся обязанностей системы.

Разница между конкретным элементом use case и сущностным может показаться едва уловимой, однако результаты этих различий оказываются значительными. Эта разница становится понятной, если привести какой-нибудь пример. Вернемся к нашей задаче получениеДенег и попробуем упростить и обобщить обычную модель use case при помощи сущностной модели, сосредоточившись на целях и задачах пользователя, а не на конкретных действиях. Для начала спросим себя, почему пользователь должен совершать такое странное действие, как вставка ценной карточки в щель какого-то автомата. Это действительно как-то глупо, особенно если учесть, что машина может просто проглотить карточку и не вернуть ее. Однако же пользователи все вставляют и вставляют в банкоматы свои карточки, делают это ежедневно и безо всяких сомнений. Почему? Потому что им нужно пройти регистрацию в системе банкомата. Делают они это просто потому, что не хотят, чтобы кто-то другой мог снять деньга с их счета. По той же причине каждый пользователь имеет собственный идентификационный номер, позволяющий системе понять, с кем она имеет

дело. Сущностный элемент use case для этого взаимодействия будет выглядеть примерно так, как в табл. 4.2.

Таблица 4.2. Сущностный элемент use case для задачи полученеДенег

Намерение пользователя	**Обязательство системы**
Регистрация в системе	
	Проверка личности
	Предложение выбора
Выбор	
	Выдача денег
Получение денег	

После проверки личности система должна предложить пользователю на выбор несколько вариантов действий, одно из которых должно считаться «обычным» действием. От 70% до 90% всех транзакций, осуществляемых банкоматами, заключаются в снятии денег с карточки, кроме того, от 70% до 90% снимаемых сумы повторяются из раза в раз для данного счета и данного клиента. Эти тенденции можно и нужно учитывать, указывая пользователю на «обычное» действие и предлагая ему либо подтвердить его, либо сменить выбор. Тем самым упрощается и ускоряется процесс проведения транзакции.

Как видите, сущностный элемент use case для той же задачи оказался гораздо меньше и проще обычного элемента use case, поскольку в него вошли только те этапы, которые существенны и которые представляют особый интерес для пользователя. Такой подход ближе к проблемно-ориентированному, а не к процедурно-ориентированному, поэтому здесь остается много открытых вопросов и возможностей по проектированию и реализации пользовательского интерфейса. Скажем, помимо идентификации при помощи карточек с магнитной лентой, существует множество других способов регистрации в системе. Например, в банкомат можно встроить систему распознавания речи, идентификации по отпечаткам пальцев, можно даже сканировать сетчатку глаза для проверки пользователя. (Интересно, что мы всегда приводили последний пример просто в методических целях, даже не задумываясь о том, что это можно технически реализовать. И вот несколько лет назад некоторые производители начали проводить эксперименты

со сканированием сетчатки для идентификации пользователей банкоматов!) Б надежно защищенных учреждениях, куда можно проникнуть, только имея специальную карточку, сканируемую лазером, автоматы, выдающие деньги служащим, могут узнавать их по этим карточкам. Еще одна возможность, никак не ограничиваемая сущностным элементом use case, заключается в использовании голосового управления.

4.4. Уровни абстракции и обобщения

Как обычные, так и сущностные элементы use case тесно связаны со сценариями. Хотя некоторые авторы определяют понятие *сценария* настолько широко, что под него подходит практически любая деятельность, связанная с повествованием, а некоторые даже используют это слово как синоним use *case*, мы полагаем, что следует обращать внимание на различие между этими понятиями.

И все-таки обычно под сценарием подразумевается конкретное и подробное описание определенной специфической последовательности событий, даже если она предназначена для представления некоторого более общего типа взаимодействия. Нильсен (Nielsen), к примеру, определяет сценарий как «замкнутое описание взаимодействия отдельного пользователя с некоторым набором вычислительных средств для достижения определенного результата при заданных условиях за определенный интервал времени. Тяжело дать более конкретнее и точное описание, чем это.

Тем не менее зачастую сценарии используются для обсуждения и описания общих шаблонов при помощи отдельных примеров. Рассмотрим, например, программу для полуавтоматической компьютерной службы технической поддержки. Пытаясь разобраться в том, каким образом система должна работать для удовлетворения потребностей пользователей, разработчик может предложить некий сценарий. Например, такой:

Сейчас 04:00. Наталья Сегаль звонит на Горячую лини» службы технической поддержки. Ншо не берет трубку, поэтому в дело вступает

автоматическая система, которая отвечает на звонок. Наталья слышит приветствие: «Добро пожаловать з систему ТехноМастер. Пожалуйста. введите свой идентификационный номер». Она вводит с помощью клавиш на телефонной трубке: «2С067Б9». после чего слыиит голосовое пеню, позволяющее выбрать один из интересующих Пунктов, и т. п.

На самом деле мы не собираемся создавать систему, которая работала бы только с пользователями по фамилии Сегаль или только в пустынные утренние часы, когда операторов нет на месте. Конкретные детали, отраженные в сценарии, просто являются ключом к пониманию общих аспектов взаимодействия. Это, кстати, одна из проблем применения сценариев в процессе создания пользовательских интерфейсов: необходимо уметь отделять общее от частного. Только в этом случае можно организовать поддержку широкого спектра взаимодействий, а не только одного варианта, описанного в сценарии. В простейших случаях выделить какие-то общие понятия и изолировать их от конкретных деталей, которые можно пропустить, не так уж трудно, но следует учитывать, что сценарии могут быть довольно длинными, преследующими не какую-то одну цель, а несколько. По мере разрастания и усложнения описаний проблемы, включения в них большого количества разных сценариев общее и частное начинают смешиваться и путаться. Во многих проектах так и не удается отделить одно от другого, в результате чего рождается проект, построенный на беспорядочной смеси обобщенных и уточненных рассуждений.

Традиционные, или «конкретные», элементы use case представляют собой сокращение или абстрагирование набора связанных сценариев. Айвар Якобсон, который ввел в обращение понятие use case, утверждает, что сценарии являются экземплярами (частными примерами) элементов use case. Таким образом, сценарий — это такое специфическое воплощение некоторой общей формы, выраженной в виде модели use case.

Одним из способов постижения фундаментальных различий между сценариями и элементами use case служит представление их в виде мнимого

кода программы. Такое сравнение, произведенное для примера с системой технической поддержки, выявит, что сценарии обычно выражаются в форме констант или литералов, например «2006789», а элементы use case будут выражены в виде переменных или символов, скажем «ID». Таким образом, несмотря на то что элементы use case моделируют отношения с конкретным пользовательским интерфейсом и, следовательно, имеют непосредственное отношение к специфическим свойствам этих пользовательских интерфейсов, изначально элементы use case являются более общей, абстрактной формой, нежели сценарии. В примере со службой технической поддержки элемент use case так или иначе связан стеле-фонным голосовым способом представления информации и вводом пользовательских данных с клавиатуры.

Проблема вот в чем: конкретные элементы use case смешивают требования с самим проектированием. По определению, они не отделяют пользовательские замыслы и цели от взаимодействия с какой-либо формой пользовательского интерфейса. Создатель конкретных элементов use case анализирует и описывает требования, выдвигаемые решаемой задачей и неявными решениями, связанными с тем, каким должен быть пользовательский интерфейс. Как ни странно, специфичность, присущая традиционным элементам use case, ведет к отсутствию ясности в соответствующей модели.

Если мы хотим максимально приблизиться к пользовательской точке зрения и сосредоточиться на понимании и описании их нужд и устремлений, нам придется использовать более абстрактную модель. Сущностный элемент use case описывает взаимодействия независимо от явных или неявных предположений, касающихся технологий или механизмов реализации. Вот как будет выглядеть сущностный элемент use case, соответствующий сценарию и конкретному элементу use case, приведенным ранее: Просьба о поноши. идентификаиия пользователя, выбор услуги и т. д.

Такой вариант описания лишен конкретных деталей, в нем нет указаний на какие-либо конкретные устройства или свойства, связанные с

пользовательским интерфейсом. Таким образом, он предоставляет возможность выбора при проектировании интерфейса для конечного пользователя. На самом деле тот же самый сущностный элемент use case мог бы описывать справочную систему, реализованную в виде Web-сайта. В этом случае пользователь просил бы о помощи, указывая URL, а не набирая номер телефона. Со своей стороны, Web-сайт для идентификации пользователей мог бы использовать cookie-файлы.

Такая форма элемента use case — это как раз и есть сущностная модель в том смысле, в котором она была ранее. Это абстрактное, не зависящее от технологии описание задачи с минимальным количеством навязываемых решений. Оно ближе любых сценариев и конкретных моделей use case к выражению пользовательских требований к создаваемой системе. Возвращаясь к проведенной при сравнении сценариев и элементов use case аналогии с программированием, можно сказать, что терминами, в которых выражаются сущностныеэлементы use case, являются суперклассы и супертипы, к которым принадлежат переменные внутри конкретных элементов use case. Поскольку мы имеем дело с сущностными элементами use case, нам предоставляется свобода выбора подходящей реализации пользовательского интерфейса. Скажем, в примере сп службой технической поддержки можно реализовать голосовую идентификацию и выбор услуги.

Сценарии, конкретные элементы use case, а также сущностные элементы — это взаимосвязанные модели задач, представляющие собой последовательные уровни абстракции, упрощения и обобщения. По мере продвижения от сценариев к сущностным элементам use case мы одновременно приближаемся к чистому и даже идеализированному выражению пользовательских потребностей и устремлений. Так как сущностные элементы лишь в упрощенном виде описывают предоставляемые системой возможности, они позволяют создавать более компактные и простые системы, которые, тем не менее, отвечают всем требованиям пользователей.

Сценарии можно также рассматривать в качестве своего рода предписаний, соответствующих одному или нескольким сущностным элементам use case. Предписание состоит из набора шагов, предпринимаемых при создании элемента use case для определенного пользовательского интерфейса. Очевидно, что для любого сущностного элемента use case можно написать множество различных предписаний, поэтому одному сущностному элементу use case могут соответствовать несколько сценариев.

Разницу между обеими разновидностями элементов use case и сценариями можно проследить также в размерах и степени детализации. Сценарии — это, по сути дела, истории, повествования, поэтому их размер может быть значительным. Чем более реалистичен сценарий, тем больше в нем красок и деталей, не все из которых действительно нужны. Примерно по тем же причинам сценарии, по крайней мере, в том виде, в котором они обычно применяются на практике для фиксации требований или управления процессом проектирования, базируются не на одном, а на нескольких сущностных элементах use case. Если сценарии обычно занимают от одной до десятков страниц, то хорошо продуманные обычные элементы use case редко занимают больше одной-двух страниц, а сущностные элементы обычно состоят из нескольких строк.

Сущностные элементы use case более компактны частично благодаря более высокому уровню абстракции, частично потому, что соответствующий подход приветствует разбиение задач на небольшие, связанные между собой части описания. Это позволяет не только ужать сами описания задач, но и реализовать принцип их повторного использования. Более подробно об этом рассказывается в следующем разделе.

5. Карты элементов use case

Элементы use case не существуют отдельно от внешнего мира. Полноценная программная система должна обеспечивать поддержку десятков, а то и сотен элементов use case, причем внутри каким-то образом связанного с ней приложений должна существовать связь между этими элементами. Отображение взаимосвязи между приложениями дает нам возможность описать общую структуру задачи, решаемой приложением и его интерфейсом. Карта элементов use case для данной задачи разбивает все функциональные возможности системы на множество взаимосвязанных сущностных элементов use case. Выделив все различающиеся и важные взаимодействия и показав отношения между ними, мы можем создать упрощенную общую модель задач, решаемых системой, и возможностей, которые она обязана предоставить.

Полноценная модель use case представляет собой множество описаний, определяющих суть всех элементов use case, и карту этих элементов, показывающую отношения между ними. Между сущностными элементами use case могут существовать отношения разных типов, включая специализацию, расширение, композицию, а также сходство. Знание этих отношений позволяет аналитику или разработчику выделить общие элементы задачи и в итоге создать более простую модель задачи.

5.1. Специализация

Некоторые элементы use case могут являться специализированными версиями других элементов. Например, в уже знакомом нам приложении «банкомат» элементы use case получениеДенег, раэмещениеСредств и эапросСостояния являются субклассами, или специализированными вариантами абстрактного класса взаимодействий, который может быть назван использован не Банкомата. Что касается отношения между элементами получениеДенег и использованиеБан комата, то его можно

охарактеризовать как классификацию, или специализацию. Такой тип отношения означает, что один элемент use case «является» («й-а») специализацией другого. В объ-ектно-ориентированном анализе и проектировании такое отношение соответствует отношению класс-подкласс.

Специализация дает нам возможность упростить общую модель use case путем отделения общих или универсальных форм взаимодействия от специфических форм, адаптированных для более узкого применения. Таким образом, у нас нет необхолимости переписывать заново самые общие паттерны применения. Достаточно написать их один раз, а затем лишь «повторно использовать» («reuse»), ссылаясь на них. Как видно из рис. 5.3, для отображения отношения специализации используется двойная стрелка. Рядом со стрелкой можно встретить подпись «is-a» или «specializes», в зависимости от контекста.

В наиболее общем случае имеется суперэлемент use case, специализируемый подэлементом use case. Он похож на абстрактный класс, то есть является элементом use case, для которого никогда не пишутся предписания. Сам по себе он не определяет какие-либо взаимодействия, реально осуществляемые пользователем. Так, клиенты банка не применяют абстрактное «использование банкомата». Они занимаются конкретными действиями: либо снимают деньги, либо размещают вклады, либо узнают баланс.

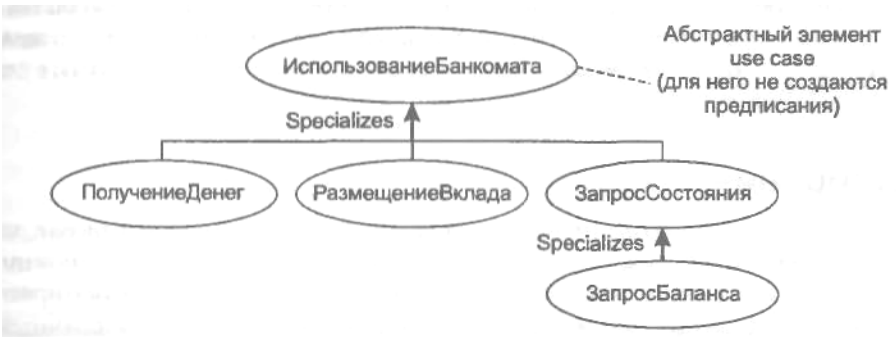

Рис. 1. Отношения специализации на карте элементов use case

В объектно-ориентированном программировании также используется отношение типа класс-подкласс, там оно часто называется наследованием и

служит средством структуризации больших систем и обеспечения повторного использования компонентов. Основное предназначение отношения специализации состоит в том, чтобы максимально упростить модель и, главное, понять, какими свойствами должен обладать пользовательский интерфейс. Поэтому построение абсолютно точной модели вовсе не является обязательной задачей разработчика. Гораздо важнее, чтобы в модели существовало разделение всех случаев применения системы на разумное количество сущностных элементов use case, осмысленных с точки зрения пользователя.

Как правило, хорошим тоном при моделировании считается ограничение отношений классификации теми случаями, когда присутствует действительно строгая связь «тип-подтип», где подэлемент представляет собой подкласс суперэлемента. Правильное использование наследования — вопрос тонкий, эта проблема активно обсуждается теми, кто занимается объектно-ориентированным проектированием. Для наших целей будет достаточно следующего: если любое предложение вида «каждый подэлемент use case — это разновидность суперэлемента use case» будет иметь смысл, значит, скорее всего, мы правильно применяем отношение специализации. Например, любой элемент полученияДенег действительно является разновидностью элемента использованияБанкомата.

Реализация пользовательского интерфейса и внутренних компонентов не обязательно в точности будет совпадать с моделью, несмотря на влияние, оказываемое картой элементов use case. Надо ведь помнить о том, что решения, принимаемые в процессе программирования, отличаются от решений, принимаемых при проектировании, на них влияет множество факторов. Известно, что практичность может пострадать из-за того, что интерфейс слишком сильно влияет на реализующий его код. Верно и обратное: буквальное моделирование пользовательским интерфейсом внутренней программной структуры ведет к тому, что внутренняя архитектура получается непрочной.

5.2. Расширение

Одной из инноваций в объектно-ориентированной программной инженерии, навеянных идеями Якобсона, стало признание расширения одним из возможных отношений между элементами use case. Говорят, что один элемент «расширяет» другой, когда он содержит вставляемые или альтернативные паттерны взаимодействия, которые войдут в расширяемый элемент. Например, при отработке элемента use case для изменения внешнего вида какой-то части экрана пользователю необходимо заниматься поиском по всей системе файла, содержащего нужную картинку или значок. Нормальное, или ожидаемое, развитие событий (обеспечивается базисом, или базисным элементом, use case) тем не менее вовсе не подразумевает поиск каких-то дополнительных графических файлов. Вот как выглядит базисный элемент use case юменениеИзображения:

Намерение пользователя	**Обязательство системы**
Запрос изменений	
	Вывод подходящих изображений
Выбор изображения	
	Вывод предварительного просмотра
Подтверждение	
	Закрытие

Понятно, что возможно множество вариантов использования (описываемых элементами use case), *в* которых может потребпваться поиск файлов с изображениями; скажем, это справедливо всегда, когда нужный файл не удается обнаружить среди предлагаемых в рамках данного варианта. Если мы создадим отдельный расширяющий элемент use case, поискИэображений, то описание базиса станет проще, а взаимодействие, отраженное в этом расширяющем элементе, сможет расширять и другие элементы. Взаимодействие, соответствующее расширяющему элементу поискИзображений, не содержится в базисе изменениеИзображений; оно может инициироваться при отрабптке базисного элемента в некоторой его точке. Это справедливо и для всех остальных элементов use case, которые расширяются элементом поискИзображений. Выделение расширений,

рассмотрение их в качестве отдельных элементов use case упрощает базисные элементы, делает их более понятными, способствует единой, унифицированной реализации всех расширений в разрабатываемом пользовательском интерфейсе, применяя таким образом принцип повторного использования.

Расширение — это удачная концепция, позволяющая значительно упростить сущностные модели use case. На карте элементов use case отношение расширения изображается в виде пунктирной линии со стрелкой и имеет подпись «extends», как показано на рис. 2. Если для расширения дается дополнительное описание, в него может быть включено примечание, показывающее, какие элементы use case расширяются.

Поиск изображений Расширяет элементы: изменениеИзображений
вставкаИзображений
открытиеФайла

Намерение пользователя	**Обязательство системы**
Повторный запрос	
	Вывод следующих изображений
[продолжать, пока не будет найден нужный файл]	
выбор изображения	Вставка изображения
	Закрытие

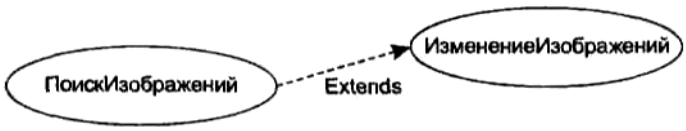

Рис. 2. Альтернативные и исключительные случаи расширения

5.3. Композиция

Элементы use case можно декомпозировать на составные части, или подэлементы, являющиеся подчиненными или включешшми паттернами взаимодействия. Отношение композиции обозначается на карте элементов

use case простой стрелкой, указывающей на подалемент use case и имеющей метку «includes», «composed of» или «uses», как показано на рис. 5.5.

Рис. 5,3. Композиция, использование одних элементов use case другими

Взаимодействие, описываемое суперэлементом use case, осуществляется при помощи взаимодействий, входящих в подэлемент или подэлементы, причем описание суперэлемента будет ссылаться на все используемые подэлементы. Например, элемент use case под названием началоПротоколированияЗадачи, созданный для про Граммы, отслеживающей ход выполнения задач, может использовать элементы авториза цияД осту па и вводПараметровЗадачи, показанные в описании элемента use case с помощью знака «>», за которым следует название используемого элемента use case. Такой способ моделирования взаимодействий позволяет разделить независимые и почти никак не связанные между собой подзадачи авторизацииДостула и вводаПараметровЗадачи. Как показано ниже, подэлемента, входящие в состав сущностного элемента use case, могут появляться как в модели намерений пользователя, так и в модели обязательств системы, в зависимости от того, что имеет больший смысл и что делает описание более естественным.

началоПротоколированияЗадачи

Намерение пользователя	**Обязательство системы**
Запрос на новую задачу	
	>авторизацияДоступа
>вводПараметровЗадачи	
	Закрытие

Отношение композиции напоминает одновременно отношения специализации и расширения. Они действительно похожи, хотя есть и существенные отличия. В отношениях соединения описание суперэлемента use case, показанного в декомпозированном виде, относится ктем

подэлементам, от которых он зависит. Подобно компьютерной программе, вызывающей стандартные подпрограммы, суперэлемент use case в явном виде ссылается на свои компоненты, без которых описание взаимодействия не было бы полным. Как при специализации, так и при расширении специализируемый/расширяемый элемент use case ничего «не знает» о подэлементах, ссылки на них отсутствуют в описании суперэлемента. Различия между этими отношениями — это вопрос видимости (области действия), вопрос того, какие элементы «знают» о других элементах или содержат явные ссылки на них. Специализации и расширения не видны тем элементам, которые они, соответственно, специализируют и расширяют, поэтому можно запросто создавать новые специализации и расширения, связанные с общим элементом use case или базисом. Разработчику модели при этом не придется вносить изменения в эти описания.

Не существует идеальных способов моделирования взаимоотношений внутри данного набора сущностных элементов use case. Одну и ту же задачу часто можно смоделировать самыми разными способами, и все они будут корректными. При этом могут использоваться отношения специализации, расширения и композиции. Как правило, композиция применяется при работе с отдельными частями и субпоследовательностями (действий), а расширение — для описания необязательных взаимодействий; специализацию лучше всего применять в тех случаях, когда существует очевидная возможность выделения подтипов. Тем не менее основным критерием всегда должен быть такой расчет: выбираем те отношения, которые самым простым образом отображают структуру задачи и в наименьшей мере ограничивают гибкость. В ранее приведенном примере с системой отслеживания хода выполнения задач элементы авторизация Доступа и вводПараметровЗадачи нужны для началаПротоколироеанияЗадачи, порядок и время их использования являются частями их определения; именно поэтому композиция предпочтительнее расшире ния: композиция позволяет получить более

простое и непосредственное представление о том, что известно о задаче. В некоторых случаях выбор может быть менее очевидным.

5.4. Сходство

Очень часто, особенно на ранних стадиях процесса моделирования, становится очевидно, что между определенными элементами use case существует какое-то отношение, однако точную его природу установить не удается. Элементы use case в приложении могут распадаться на смысловые или логические множества, например на группы элементов use case, каким-то образом связанные с оценкой затрат на путешествие. Мы используем концепцию сходства для описания как раз таких неоднозначных взаимоотношений, которые тяжело точно определить Сходство представляет собой очевидное, но не специфицированное отношение между элементами use case. Сходство моделируется путем группировки элементов use case в логические кластеры, которая производится в зависимости от степени подобия. Сходные элементы мы размещаем вместе на карте элементов use case. Степень сходства может определять то, насколько далеко мы расположим элементы на карте. Может быть, они будут даже накладываться друг на друга, а может быть, между ними будет сохраняться какая-то дистанция. Например, можно считать, что элементы получен иеДенег и размещениеВклада более схожи между собой, нежели с элементом эапросСостояния. При этом мы можем и не быть уверены в том, что именно подразумевает это сходство. Для этих трех элементов мы можем создать модель, показанную на рис. 5.4. Если мы считаем нужным, то можем даже наложить графические изображения элементов use case, как это случилось с элементами размещениеВклада и переводДенег, Можем соединить элементы и пунктирной линией без стрелки. Рядом с линией можем сделать надпись, отражающую наши познания относительно природы отношений между элементами use case. Зачастую эти по-анвния выражаются одним лишь словом: «напоминает» («resembles»).

Рис. 5,4. Сходство элементов use case

Отношение подобия особенно важно на ранних стадиях жизненного цикла разработки ПО, когда аналитик/проектировщик еще не вполне определился и не очень хочет окончательно зафиксировать свою модель, потому что многие детали еще не понятны или считаются несущественными. Тем не менее отношение сходства может служить ценным источником информации, ведущим к созданию правильной архитектуры интерфейса. За счет чего это достигается? В частности, за счет поддержки структурного принципа, подразумевающего расположение подобных элементов интерфейса недалеко друг от друга и разделение не связанных между собой элементов.

Иногда два элемента use case, выражающие разные намерения пользователя, могут оказаться почти идентичными или эквивалентными, когда дело доходит до моделирования задач, которые описываются с их помощью. Мы соединяем такие элементы use case двойной прерывистой линией и помечаем их как «эквивалентные» («equivalent»), или «идентичные» («identical»). Таким образом, карта элементов use case с точки зрения пользователя остается полноценной и одновременно говорит разработчикам о том, что не нужно создавать для данных элементов отдельные реализации.

5.5. Центральные элементы use case

Прежде чем считать процесс создания сущностной модели use case завершенным, следует определить один или несколько центральных элементов use case. Центральные элементы use case так называются потому, что именно они находятся в центре внимания при организации всего пользовательского интерфейса или его части. Центральные элементы use case

для всей системы вбирают в себя все самые важные, представительные, одним словом, центральные варианты использования. В большинстве случаев центральные элементы use case будут осуществлять поддержку центральных пользовательских ролей, определенных в ролевой модели.

Например, если взять уже однажды рассмотренный нами апплет, расширяющий возможности клавиатуры, то роли СлучайныйОбычныйНабирательСимволов и Слу-чайныйПереводчик можно считать центральными. Элементы use case, поддерживающие эти роли, могут включать тесно связанные между собой элементы вставкаСимвола и вставкаФразы, которые, в своюочередь, можно рассматривать как центральные элементы use case. На диаграмме use case центральные элементы use case изображаются двойными эллипсами.

Помимо центральных элементов use case для приложения в целом, они могут быть определены для любой подсистемы или другой структурной единицы в составе приложения. Эти центральные элементы помогают организовать проектирование соответствующих частей системы.

Элементы use case могут быть назначены центральными исходя из разных соображений. Среди таких соображений могут быть значимость для пользователя и даже финанспвые риски. Не существует какого-либо единственно верного способа выбора центральных ролей или центральных элементов use case. Тут как при фотосъемке; если мы наводим фокус на что-то одно, то что-то другое обязательно получается расплывчатым. Центральные элементы use case служат отправной точкой для разработки целесообразной архитектуры пользовательских интерфейсов. Но надо отличать отправную точку процесса от самого процесса. Если внимательно слепить за деталями, вовремя производить улучшения и доработки, то даже неверный выбор центральных элементов use case не приведет к каким-либо проблемам.

5.6. Создание сущностных моделей use case

5.6.1. Идентификация элементов use case

Логичнее всего начинать идентификацию элементов use case с модели пользовательских ролей. Взаимоотношения с системой, воплощенные в ролях пользователей, потребности и намерения пользователей — все это обычно пересекается. Реальные пользователи, знакомые с теми или иными ролями, все те, кто будет играть эти роли при работе с системой, также могут помогать при создании моделей.

Мы изучаем пользовательские роли и в результате этого создаем наборы потенциальных элементов use case. Сосредоточившись на главном предназначении системы, на намерениях пользователей, мы начинаем задавать вопросы, касающиеся каждой роли.

1. Какие задачи будут пытаться решать пользователи, играющие данную роль?
2. Что должны уметь делать пользователи, чтобы играть данную роль?
3. Какие возможности должна предоставлять система, чтобы пользователь мог решить любые задачи, присущие данной роли?

Якобсон дополнил этот список еще несколькими вопросами, касающимися каждой роли.

1. Каковы *основные* задачи, решаемые пользователями, выступающими в данной роли?
2. Какую информацию пользователям в этой роли необходимо анализировать, создавать или изменять?
3. Что потребуется пользователям в этой роли, чтобы они были в курсе происходящего в системе?
4. О чем пользователи в этой роли должны информировать систему?

Первый из этих вопросов может показаться расплывчатым, зато все остальные охватывают уже знакомый круг проблем — в основном эта стандартные задачи, возникающие при традиционной обработке информации.

Процесс создания полноценной сущностной модели use case может проходить по-разному. Многие разработчики предпочитают применять метод мозгового штурма, создавая максимально большой список потенциальных элементов use case и не особо заботясь о точных деталях или определениях. Все сущностные элементы use case помечаются при помощи собственных названий и/или идентификаторов, соответствующих их основному предназначению. Этот предварительный список кандидатов затем тщательно изучается и в случае необходимости аисращается за счет исключения пересекающихся и слияния похожих или взаимосвязанных элементов.

После этого для элементов use case создаются описания, определяются отношения меаду ними. Некоторые разработчики предпочитают начинать с одного потенциального элемента use case, доводить его до совершенства, создавать для него полноценное описание, а затем уже переходить к другим элементам.

Метод мозгового штурма при создании списка потенциальных элементов use case имеет одно важное преимущество: он позволяет быстро определить все возможные варианты использования, поддерживаемые системой. Элемент use case в этом случае быстро выявляется, ему дается простое имя, он снабжается кратким описанием применения, а кроме всего этого мозговой штурм способствует формированию более абстрактного и общего взгляда на задачи, связанные с сущностным моделированием. Зато если начать с описания одного конкретного элемента use case, можно сразу сосредоточиться на анализе реальных проблем. В любом случае, если наступает ступор, если процесс увязает в решении какого-то одного вопроса, имеет смысл сменить подход. Если вы уже выдохлись, придумывая список кандидатов, просто начните детализировать описания для каждого элемента

use case. Если, наоборот, вы устали размышлять о деталях описания, подумайте, какие еще элементы use case можно придумать для данной системы.

5.6.2. Пользователи и элементы use case

ПО, применяемой в той организации, где вы работаете, процесс создания модели задач, как и ролевой модели, будет в большей или в меньшей степени включать в себя взаимодействие с пользователями. Некоторые 1руппы программистов считают весьма производительным подход, при котором представители пользователей участвуют в мозговом штурме или даже в создании описаний элементов use case. Другие предпочитают начинать с неформального общения с пользователями, а реальную работу по моделированию выполнять самостоятельно.

В любом случае, всегда полезно сверять модели use case с пользователями будущей системы. Описания элементов use case и сущностных моделей use case легко воспринимаются пользователями, поэтому проблем с пониманием между разработчиками и заказчиками на этом этапе возникать не должно. Для пользователей описание, связанное с элементом use case, — это своего рода диалог, организованный в виде двух текстовых столбцов. Для того чтобы вникнуть в этот диалог, не требуется никакой предварительной подготовки. Если разработчики опираются на дух, присущий моделям use case, и используют язык, характерный для данной прикладной области и понятный пользователям, то у последних не возникнет никаких проблем с чтением и конструктивной критикой этих повествований. Некоторые пользователи склонны считать описания элементов use case чрезмерно упрощенными и лишенными многих важных Деталей, поэтому разработчик должен четко объяснить им, что целью в данном случае является моделирование взаимодействий, а не реализующих их механизмов. Это одна из причин, по которым разработчики предпочитают общаться с пользователями посредством сценариев, которые затем переделываются в

сущностные элементы use case путем абстрагирования и обобщения. Следующий вопрос заключается в том, сверять ли с пользователями разработанное описание сущностной модели. Эти описания служат в основном руководящими указаниями при проектировании пользовательского интерфейса, однако можно поделиться сущностными моделями use case с пользователями и получить в результате обратную связь на ранних этапах разработки, что позволяет оценить завершенность модели и ее точность.

5.6.3. Создание описаний элементов use case

Процесс создания описания элемента use case начинается с выяснения основного предназначения этого элемента или пользовательских намерений, воплощенных в данном взаимодействии. Итак, чем же занимаются пользователи? Чего они пытаются добиться, отрабатывая тот или иной вариант использования? Зачем они этого добиваются?

Посче определения основного предназначения элемента use case ему присваивается простое имя, служащее намеком на связанную с ним целесообразную, осмысленную деятельность. Отглагольные существительные, глаголы, выражающие продолжительное действие, направленное непосредственно на некоторый объект, отлично подходят в качестве названий для сущностных элементов use case. Например:

поискКлиента

проверкаЗакаэа

вставкаМатематическогоСимвола

Если цели или намерения пользователя выражены не очень четко или не очевидны из названия элемента use case, в заголовок описания можно добавить специальный пункт предназначение, описывающий и детализирующий назначение элемента use case с точки зрения пользователя. Важно не зацикливаться на подборе названия, особенно если речь идет о группе разработчиков. При этом не имеет никакого значения присутствие или отсутствие пользователя. Долгие споры о том, как следует назвать

элемент use case, никогда не бывают продуктивными и только отвлекают от сути процесса. Абсолютно правильное название не так уж важно, гораздо важнее написать простое, понятное описание. Но, конечно, никуда не годное имя может привести к неверному толкованию варианта использования, К тому же это усложняет задачу выявления похожих или даже одинаковых элементов use case. Имена, отражающие самую суть элементов use case, помогают определять отношения между элементами, упрощают процесс общения с пользователями. Названия, которые слишком сильно опираются на тяжелые термины из словаря по программированию или обработке информации, смущают и раздражают пользователей, а у разработчиков создают ложное ощущение, будто они что-то понимают, хотя на самом деле это не так. Приучая разработчиков мыслить общепринятыми терминами, мы обычно запрещаем им пользоваться какими-либо техническими понятиями и названиями, заимствованными из программирования, заставляем аналитиков переводить свои высказывания на человеческий язык, понятный пользователям и используемый в интерфейсах.

Основой сущностного моделирования является постоянное возвращение к вопросу назначения всех элементов. Думая о каждом действии пользователя, мы должны спрашивать себя; для чего оно нужно? Почему пользователь совершает его? Почему вообще кому-то может понадобиться совершать его? В чем заинтересован пользователь, каковы его намерения, чего он пытается добиться, обращаясь к этому действию? Всегда нужно стараться использовать тот язык, который помогает сосредоточиться на цели, а не на средствах взаимодействия. Кроме того, мы ведь упрощаем и обобщаем, не надо об этом забывать.

Описание сущностного элемента use case по возможности следует выражать в терминах пользователей и прикладной области. Избегайте конструкций, заимствованных из программирования, а также жаргона компьютерщиков. Некоторые разработчики пытаются использовать разные виды символического кода или «структурированного языка» для выражения

в описаниях итераций и условий. К сожалению, такая практика приводит только лишь к тому, что внимание переключается с насущных проблем непонятно на что, в лучшем случае — на проблемы программной реализации. Описания, напоминающие компьютерные программы, страшно пугают обычных, не искушенных в профаммировании пользователей, они выглядят гораздо непонятнее и загадочнее описаний с применением обычной лексики. Применение непривычной лексики нарушает нормальный диалог с пользователями, усложняет проверку корректности и полноценности сущностных моделей use case.

В общем, мы считаем, что неформальный язык, даже если он иногда и неточен, лучше всего подходит для описаний сущностных элементов use case. Вместо того чтобы писать якобы программистские фразы типа do-while и else-if, аккуратно включая во все выражения операторные скобки begin-end, мы предпочитаем такие выражения, как «продолжать е том же духе, пока не надоест» или «попробовать при желании». Покуда пользователь понимает написанное и покуда разработчики и пользователи интерпретируют описания одинаковым образом, приемлемы любые фразы и обороты.

Некоторым кажется, что мыслить в терминах абстрактных взаимодействий и каких-то главных устремлений, не опираясь на конкретные действия, очень трудно. Обойти это препятствие можно, если начать с выписывания одного-двух сценариев, их обобщения и приведения к более абстрактному виду. Может быть, придется делать это даже в несколько этапов, прежде чем будет создано достаточно сущностное описание. Многим, впрочем, удобнее изначально мыслить общими понятиями. Тут «правильного» пути нет, каждый выбирает для себя.

В любом случае, одним из важнейших этапов, возможно, самым важным во всем процессе практичного проектирования, является возвращение назад и внимательное изучение того, что уже сделано. Может быть, стоит попытаться еще больше упростить и обобщить написанное. Этот шаг необходимо повторять снова и снова, только тогда удастся создать более

дешевую и удачную систему. То, что разработчик делает при создании сущностных описаний, задает некий нижний предел простоты и компактности будущего интерфейса.

6. Тесты, измерения и метрики

6.1. Измерение качества

Метрики практичности — это, по сути дела, метрики качества ПО, количественный или оценочный показатель некоторых факторов или аспектов качества программной продукции. У этих метрик долгая и славная история применения в программной инженерии. Метрики практичности — это относительно новое направление, которое можно считать продолжением сущностного моделирования, помогающего разработчикам находить ответы на некоторые важные вопросы. Метрики нельзя считать панацеей от всех проблем практичности программного обеспечения, однако, наряду с повышением качества моделей, гибкости и эффективности методов проектирования, метрики предоставляют разработчикам дополнительные средства поиска оптимальных решений.

6.2. Метрики

Все метрики можно разбить на три больших категории:

– метрики предпочтений — служат для количественного измерения субъективных пользовательских оценок и предпочтений;

– метрики производительности — с их помощью измеряют реальный КПД программного обеспечения;

– предсказывающие метрики — также называются проектными метриками; служат для оценки качества проектных решений или прототипов.

6.3. Метрики предпочтений

Один из наиболее простых и популярных методов оценки практичности заключается в использовании метрик предпочтений. Технология такова: берете несколько ничего не подозревающих субъектов — пользователей или потенциальных пользователей — и спрашиваете их, что они думают о данной системе. Естественно, у субъективной пользовательской оценки есть как

плюсы, так и минусы. Плюсы — в относительной дешевизне и простоте получения информации, минусы — в том, что не всегда именно эта информация требуется разработчикам. Оценка пользователей не только субъективна сама по себе, но и пользовательские предпочтения весьма относительно коррелируют с реальной простотой использования и другими стандартными показателями практичности. Пользователи будут говорить вам про невиданное удобство использования и указывать при этом на запутанные или неэффективные системы. Попробуйте спросить, что они думают по поводу Web-сайгов, и в их ответе вы найдете подтверждение этим словам. Тем не менее субъективные впечатления конечных потребителей продукта игнорировать нельзя. Удовлетворение пользователя — это, несомненно, одна из составляющих практичности и важный фактор успеха на рынке. Ведь в самом деле, удовлетворение пользователя лучше подскажет, какой продукт «пойдет» на рынке, чем практичность сама по себе.

Многие разработчики пытаются создавать собственные схемы оценки пользовательских предпочтений. Например, можно показывать пользователям два экранных снимка (или карандашных наброска) и спрашивать, какой из них нравится больше. Более систематический подход подразумевает предварительное создание списка вопросов, при ответе на которые можно пользоваться пятибалльной шкалой. Такие собственноручно созданные списки особенно полезны в тех случаях, когда необходимо решать чрезвычайно специфические проблемы или когда речь идет о приложении для какой-нибудь необычной предметной области. Например, команда проектировщиков может захотеть узнать предпочтения пользователей телефонной справочной службы относительно того, как они хотят сообщать свои ответы: устно или нажимая клавиши на трубке. Или же можно показать потенциальным пользователям виртуальных шлемов несколько экранных снимков, отпечатанных на пленке, и спросить, какую цветовую гамму они предпочитают.

Создание анкет для пользователей, как и проектирование пользовательских интерфейсов, — это искусство и наука одновременно. И здесь также есть множество хитростей. Субъективные оценки, сделанные пользователями, могут быть весьма сложными, со множеством факторов, которые необходимо тщательно отделять друг от друга и, разумеется, изучать. Стандартные, проверенные бланки или анкеты обладают реальными преимуществами. Они надежнее единичных опросов, их адекватность обычно уже проверена временем, изучена, и доказательством тому может служить их широкое применение. Кроме того, они дают возможность сравнить впечатления пользователей со стандартными оценками других программных систем.

6.4. Метрики производительности

Некоторые вопросы удается решить, только поработав с реально функционирующей системой; соответственно, в задачи тестирования практичности входит создание условий, достаточно реалистично моделирующих такую работу. Мера «достаточности» в данном случае определяется мерой искажения ответов пользователей. Метрики производительности служат показателями того, как пользователи выполняют задания с помощью системы в лабораториях тестирования практичности либо на своих рабочих местах. Можно измерять множество аспектов производительности, например время, затрачиваемое на решение задачи или нескольких задач, количество ошибок, частоту обращения к справочной системе.

6.5. Предсказывающие метрики

Метрики проектирования являются объективными показателями качества, которые можно получить, используя элементы проекта, такие как визуальный дизайн экранных решений. Эти метрики называются предсказывающими, поскольку они позволяют предположить или предсказать некоторые аспекты реальной производительности уже

построенной системы. Корректные метрики проектирования будут достаточно точно коррелировать с реальной простотой использования, эффективностью, уровнем ошибок и другими показателями прикладной практичности. Разработчики ПО могут рассматривать метрики проектирования как альтернативу субъективным пользовательским оценкам или как тестирование практичности, проводимое уже после создания функционального прототипа или системы. Большим преимуществом метрик проектирования является то, что они позволяют быстро и дешево оценивать и сравнивать разные проекты. При этом не требуется заранее специально для тестирования разрабатывать систему, модель или рабочий прототип.

6.6. Метрики проектирования пользовательского интерфейса

6.6.1. Объекты измерений

Метрики проектирования для оценки пользовательских интерфейсов могут существовать в следующих вариантах:

– структурные метрики, базирующиеся на поверхностных свойствах;

– семантические метрики, чувствительные к содержательному наполнению (контенту);

– процедурные метрики, чувствительные к решаемым задачам.

Легче всего вычисляются показатели, объединенные в группу под названием **структурные** метрики. Они базируются на тех поверхностных свойствах архитектурных решений пользовательского интерфейса, которые легко сопоставитьс конкретными цифрами. Существует ряд уже испробованных структурных метрик, среди которых:

– количество визуальных компонентов или элементов управления на экране или в диалоговом окне;

– количество и распределение пустого пространства между элементами;

– взаиморасположение элементов;

– число экранов или диалоговых окон, напрямую достижимых из данного экрана или диалогового окна;

– диаметр (то есть максимальная длина) цепочки переходов между экранами или диалоговыми окнами.

Многие из простейших вышеперечисленных структурных метрик представляют собой чрезмерно упрощенную попытку измерения сложности. Казалось бы, вполне логично, например, предположение о том, что экран с большим количеством элементов сложнее экрана с небольшим количеством элементов и, следовательно, тяжелее в использовании (при прочих равных условиях).

К сожалению, многие простые структурные метрики не отличаются хорошей теоретической базой, которая привязывала бы их к реальной практичности конечных продуктов. Для разработчиков, к тому же, важен еще и тот факт, что структурные метрики не решают повседневные вопросы проектирования. Надо сказать, люди, проектирующие пользовательские интерфейсы, обычно имеют дело с более неприятными и значительными проблемами, нежели количество элементов управления в диалоговом окне или объем окружающего их пустого пространства.

В отличие от простых структурных метрик — семантические метрики позволяют измерить показатели, зависящие от контента и природы компонентов интерфейса: их функциональных возможностей, назначения и работы. Семантические метрики служат для оценки тех аспектов интерфейса, которые зависят от концепций и действий, представляемых компонентами этого интерфейса, а также оттого, как пользователи воспринимают компоненты и их взаимосвязь.

Процедурные метрики чувствительны к задачам, решаемым интерфейсам. Аспекты, которые удается измерить с их помощью, связаны с реальными задачами или сценариями исследуемого интерфейса. Процедурные метрики позволяют понять соотношение различных задач и данного проектного решения в терминах содержательного наполнения

(контента) и организации. В целом, можно ожидать от метрик, зависящих от контента и задач, более полезных результатов, чем от простых структурных метрик. В принципе, данная метрика может быть чувствительной как к контенту, так и к задачам, однако на практике чаще всего требуются раздельные метрики.

6.6.2. Критерии измерений

Наиболее полезными для практикующих разработчиков при решении каждодневных вопросов проектирования пользовательских интерфейсов окалываются стабильные, простые и понятные метрики. Настоящие, хорошие метрики должны:

– обладать легко вычисляемыми и интерпретируемыми показателями;

– быть удобными для оценки бумажных прототипов и проектных моделей;

– иметь прочное обоснование и простой концептуальный базис;

– быть достаточно чувствительными, чтобы реагировать на разные решения;

– подсказывать возможные проектные решения;

– эффективно предсказывать реальную практичность будущей системы,

– непосредственно выявлять относительное качество проекта.

Сильный и простой концептуальный базис означает, что разработчики должны иметь возможность сразу же понять обоснование метрики, представлять себе, почему (с точки зрения этой метрики) одни решения лучше других. Результаты кажутся более убедительными, когда за цифрами стоит что-то более основательное.

Хорошая концептуальная база упрощает понимание разработчиками того, как разные значения показателей влияют на изменение проектов.

В идеале и сами метрики, и процесс проведения измерений должны указывать разработчикам, как сделать проекты лучше. При этом метрики, позволяющие работать непосредственно с визуальным дизайном,

проектными моделями или не-функционирующими визуальными прототипами, полезнее, нежели те метрики, для подсчета которых требуются полностью или частично работающие системы.

Один из вопросов, столь часто интересующих разработчиков, состоит в том, насколько проект хорош в некотором абсолютном измерении. Абсолютная оценка бывает полезна при принятии решений о проведении дальнейших улучшений (тестирования) или об отказе от них. Метрика, предоставляющая набор произвольных показателей, позволяет сравнить разные решения, но не отвечает на вопрос о том, насколько хорош или плох тот или иной проект сам по себе. Может быть, он близок к совершенству, а может быть, существует еще множество возможностей для улучшений? Метрики, построенные на принципе сравнения данного решения с идеальным или позволяющие оценить, насколько оно подходит для выполнения конкретных задач, несомненно, более полезны для разработчиков.

Эффективные метрики должны, разумеется, достаточно точно предсказывать важные аспекты практичности программного обеспечения в его конкретных применениях — например, время выполнения задач, время изучения интерфейса, уровень ошибок. Метрики проектирования пользовательских интерфейсов, кроме того, должны быть достаточно чувствительными, чтобы различать похожие решения, обладающие разной конечной практичностью. В идеале сама метрика или способ ее вычисления должны предоставлять информацию, подсказывающую пути улучшения существующего проекта.

7. Квантификация

Существует множество методов количественного анализа элементов интерфейса. Однако ясное руководство о том, как использовать эти методы, можно встретить редко. В данном разделе будет дано простое описание и подробные примеры модели GOMS, разработанной Кардом, Мораном и Ньюэллом, а также критерий эффективности Раскина, закон Хика и закон Фитса.

7.1. Количественный анализ интерфейса

Многие количественные и эвристические методы используются для анализа и изучения интерфейсов. Эти методы составляют значительную часть содержания большинства книг, посвященных этой теме, включая и те, которые указаны в библиографическом списке за такими авторами, как Шнейдерман (Shneiderman), Норман (Norman) и Мэйхью (Mayhew). Например, с помощью пассивного наблюдения за тестированием нового интерфейса с участием нескольких добровольцев опытный разработчик интерфейсов может узнать столько же ценной информации, сколько можно получить с помощью любого метода количественного анализа. Здесь необходимо сосредоточиться на количественных методах не для того, чтобы принизить значение качественных методов, но скорее для того, чтобы найти между ними баланс и показать ценность численных и эмпирических методов, которые не являются широко известными. Количественные методы часто могут свести спорные вопросы к простым вычислениям. Еще одним, более важным преимуществом этих методов является то, что они помогают нам понять важнейшие аспекты взаимодействия человека с машиной.

Одним из лучших подходов к количественному анализу моделей интерфейсов является классическая модель GOMS — «правила для целей, объектов, методов и выделения» (the model of goals, objects, methods, and selection rules), которая впервые привлекла к себе внимание в 80-х годах.

Моделирование GOMS позволяет предсказать, сколько времени потребуется опытному пользователю на выполнение конкретной операции при использовании данной модели интерфейса. После обсуждения модели GOMS мы рассмотрим количественные методы оценки производительности интерфейсов, скорости движения курсора и времени, необходимого для принятия решения.

7.2. Модель скорости печати GOMS

Здесь мы обсудим только один простейший, но довольно ценный аспект метода GOMS — модель, основанная на оценке скорости печати. Разработчики, которые знакомы с методом GOMS, редко проводят детальный и формальный анализ модели интерфейса. Отчасти это происходит из-за того, что основы GOMS и других количественных методов известны им настолько, что они изначально руководствуются этими методами в процессе разработки. К формальному анализу, конечно, прибегают в случаях, когда необходимо выбрать один из двух вариантов разработки, когда даже небольшие различия в скорости могут давать большой экономический и психологический эффект. Иногда разработчики пользуются поражающими своей точностью расширенными моделями GOMS, как, например, анализ с использованием метода критического пути GOMS (critical-path method GOMS, CPM-GOMS) или версия, называемая естественным языком GOMS (natural GOMS language, NGOMSL), в которой учитывается поведение неопытного пользователя, например время, необходимое ему для обучения. С помощью этих методов можно, например, предсказать, сколько времени понадобится пользователю для выполнения некоторого набора действий при использовании данного интерфейса с абсолютной погрешностью менее 5%. В расширенных моделях почти все оценки не выходят за пределы стандартного отклонения, принятого для измеренных значений времени. Для вопросов, которые вызывают жаркие споры и по поводу которых авторитетные разработчики зачастую

высказывают совершенно разные мнения, полезно вооружиться количественными методами, имеющими теоретическое обоснование и получившими экспериментальную апробацию.

7.3. Временные интервалы в интерфейсе

Разработчики модели GOMS во время ее создания заметили, что время, требующееся для выполнения какой-то задачи системой «пользователь — компьютер», является суммой всех временных интервалов, которые потребовались системе на выполнение последовательности элементарных жестов, составляющих данную задачу. Хотя для разных пользователей время выполнения того или иного жеста может сильно отличаться, исследователи обнаружили, что для большей части *сравнительного* анализа задач, включающих использование клавиатуры и графического устройства ввода, вместо проведения измерений для каждого отдельного пользователя можно применить набор стандартных интервалов. С помощью тщательных лабораторных исследований был получен набор временных интервалов, требуемых для выполнения различных жестов. Ниже приводится оригинальная номенклатура, в которой каждый интервал обозначен одной буквой.

K = 0.2 с	Нажатие клавиши. Время, необходимое для того, чтобы нажать клавишу.
P = 1.1 с	Указание. Время, необходимое пользователю для того, чтобы указать на какую-то позицию на экране монитора.
H = 0.4 с	Перемещение. Время, необходимое пользователю для того, чтобы переместить руку с клавиатуры на ГУВ или с ГУВ на клавиатуру.
M = 1.35 с	Ментальная подготовка. Время, необходимое пользователю для того, чтобы умственно подготовиться к следующему шагу.
R	Ответ. Время, в течение которого пользователь должен ожидать ответ компьютера.

На практике указанные значения могут варьироваться в широких пределах. Для опытного пользователя, способного печатать со скоростью 135 слов/мин., значение *K* может составлять 0.08 с, для обычного пользователя, имеющего скорость 55 слов/мин., — 0.2 с, для среднего неопытного

пользователя, имеющего скорость 40 слов/мин., — 0.28 с, а для начинающего — 1.2 с. Нельзя сказать, что скорость набора не зависит от того, что именно набирается. Для того чтобы набрать одну букву из группы случайно взятых букв, большинству людей требуется около 0.5 с. Если же это какой-то запутанный код (например, адрес электронной почты), то у большинства людей скорость набора составит около 0.75 символов в секунду. Значение ***K*** включает в себя и то время, которое необходимо пользователю для исправления сразу замеченных ошибок. Клавиша <Shift> считается за отдельное нажатие.

Широкая изменяемость каждой из представленных мер объясняет, почему эта упрощенная модель не может использоваться для получения абсолютных временных значений с какой-либо степенью точности. Тем не менее, с помощью типичных значений мы можем сделать правильную *сравнительную оценку* между какими-то двумя интерфейсами по уровню эффективности их использования. Если оцениваются сложные интерфейсы, включающие пересекающиеся временные зависимости, или если должны быть с точностью достигнуты определенные временные интервалы, то следует применять более сложные модели (например, CPM-GOMS), которые не рассматриваются в этой книге.

Длительность ответа, поступающего от компьютера, ***R***, может оказывать неожиданный эффект на действия пользователя. Если при использовании какого-то управляющего элемента на экране монитора в течение приблизительно 250 мс ничего не возникает, пользователь, скорее всего, может почувствовать беспокойство, решит сделать еще одну попытку или подумает, что система неисправна.

Нельзя сделать такой продукт, который мог бы завершать все операции за время, не превышающее времени реакции пользователя, но можно сделать такие интерфейсы, в которых в течение этого времени всегда бы выдавалось сообщение о том, что информация от пользователя принята и правильно распознана. В противном случае действия пользователя во время задержки —

чаще всего просто молотьба по клавиатуре с целью получить хоть какой-то ответ — могут привести к нежелательным реакциям со стороны системы, приводя тем самым к еще большей задержке или даже повреждению содержания.

Если задержки неизбежны, важно, чтобы в интерфейсе была предусмотрена сообщающая о них обратная связь, — например, можно использовать индикатор хода выполнения задачи, отражающий время, оставшееся до конца операции. Если неизвестно, сколько именно времени займет операция, так и скажите об этом пользователю! Нельзя лгать пользователю или вводить его в заблуждение.

7.4. Расчеты по модели GOMS

Вычисления времени, необходимого на выполнение того или иного действия (например, «переместить руку с графического устройства ввода на клавиатуру и набрать букву»), с помощью модели GOMS начинаются с перечисления операций из списка жестов модели GOMS, которые составляют это действие (в приведенном примере это *H K*). Перечисление движений (*K*, *P* и *H*) — это довольно простая часть модели GOMS. Более сложным, например, в модели скорости печати GOMS, является определение точек, в которых пользователь остановится, чтобы выполнить бессознательную ментальную операцию, — интервалы ментальной подготовки, которые обозначаются символом *M*. Основные правила, позволяющие определить, в какие моменты будут проходить ментальные операции, представлены в табл. 7.1.

Таблица 7.1. Расстановка ментальных операций

Правило 0 Начальная расстановка операторов M	Операторы M следует устанавливать перед всеми операторами K (нажатие клавиши), а также перед всеми операторами P (указание с помощью ГУВ), предназначенными для выбора команд; но перед операторами P, предназначенными для указания на аргументы этих команд, ставить оператор M не следует.
Правило 1 Удаление ожидаемых	Если оператор, следующий за оператором M, является полностью ожидаемым с точки зрения оператора,

операторов M	предшествующего M, то этот оператор M может быть удален. Например, если вы перемещаете ГУВ с намерением нажать его кнопку по достижении цели движения, то в соответствии с этим правилом следует удалить оператор M, устанавливаемый по правилу 0. В этом случае последовательность P M K превращается в P K.
Правило 2 Удаление операторов M внутри когнитивных единиц	Если строка вида M K M K M K... принадлежит когнитивной единице, то следует удалить все операторы M, кроме первого. Когнитивной единицей является непрерывная последовательность вводимых символов, которые могут образовывать название команды или аргумент. Например Y, перемещать, Елена Троянская или 4564.23 являются примерами когнитивных единиц.
Правило 3 Удаление операторов M перед последовательными разделителями	Если оператор K означает лишний разделитель, стоящий в конце когнитивной единицы (например, разделитель команды, следующий сразу за разделителем аргумента этой команды), то следует удалить оператор M, стоящий перед ним.
Правило 4 Удаление операторов M, которые являются прерывателями команд	Если оператор K является разделителем, стоящим после постоянной строки (например, название команды или любая последовательность символов, которая каждый раз вводится в неизменном виде), то следует удалить оператор M, стоящий перед ним. (Добавление разделителя станет привычным действием, и поэтому разделитель станет частью строки и не будет требовать специального оператора M.) Но если оператор K является разделителем для строки аргументов или любой другой изменяемой строки, то оператор M следует сохранить перед ним.
Правило 5 Удаление перекрывающих операторов M	Любую часть оператора M, которая перекрывает оператор R, означающий задержку, связанную с ожиданием ответа компьютера, учитывать не следует.

Кроме того, отметим, что в этих правилах под строкой будет пониматься некоторая последовательность символов. Разделителем будет считаться символ, которым обозначено начало или конец значимого фрагмента текста, такого как, например, слово естественного языка или телефонный номер. Например, пробелы являются разделителями для большинства слов. Точка является наиболее распространенным разделителем, который используется в конце предложений. Скобки используются для ограничения пояснений и замечаний и т.д. Операторами являются K, P и H. Если для выполнения команды требуется дополнительная информация (как, например, в случае когда для установки будильника пользователю требуется указать время его включения), эта информация называется аргументом данной команды.

7.5. Примеры расчетов по модели GOMS

Разработка интерфейса обычно начинается с определения задачи или набора задач, для которых продукт предназначен. Суть задачи, а также средства, имеющиеся для реализации ее решения, часто формулируют в виде требования или спецификации.

Требования

Пусть имеется лаборант, который работает на компьютере — печатает отчеты. Иногда его отвлекают экспериментаторы, находящиеся в этой же комнате, чтобы попросить перевести температурные показания из шкалы Фаренгейта в шкалу Цельсия или наоборот. Например, лаборанту могут сказать: «Переведи, пожалуйста, 302.25 градуса по шкале Фаренгейта в градусы по шкале Цельсия». Значение температуры лаборантможет ввести только с помощью клавиатуры или ГУВ. Голосовые или другие средства ввода отсутствуют. Просьбы о переводе из одной шкалы в другую поступают приблизительно с равной вероятностью. Приблизительно 25% значений — отрицательные. 10% значений являются целочисленными (например, 37°). Результат перевода из одной шкалы в другую должен отражаться на экране монитора. Другие средства вывода результатов не используются. Лаборант читает вслух экспериментатору полученное значение. Вводимые и выводимые числовые значения температур могут иметь до десяти цифр с каждой стороны от десятичного разделителя.

При разработке интерфейса для системы, с помощью которой лаборант сможет выполнять такие просьбы, следует минимизировать время, необходимое для перевода из одной шкалы в другую. Скорость и точность операций должны быть максимальными. Рабочая площадь экрана не ограничена. Окно или область экрана, предназначенная для перевода температурных значений, является постоянно активным и готово к вводу данных с помощью клавиатуры или ГУВ. То, каким образом лаборант сможет вернуться к выполнению его основной работы, не учитывается. Задача считается выполненной с получением результата перевода.

Для оценки требуемого лаборанту времени исходите из среднего временного значения на введение четырех символов, включая десятичную запятую. Также, из соображений простоты, будем считать, что лаборант вводит все символы без ошибок, и поэтому средства выявления ошибок и сообщения о них не требуются.

7.5.1. Интерфейс для программы: вариант 1. Диалоговое окно

Инструкции в диалоговом окне (рис. 7.1) довольно просты. На их основе можно описать метод действий, который должен использовать лаборант в терминах жестов модели GOMS. Запись по модели GOMS будет представлена последовательно по мере того, как будут добавляться новые жесты.

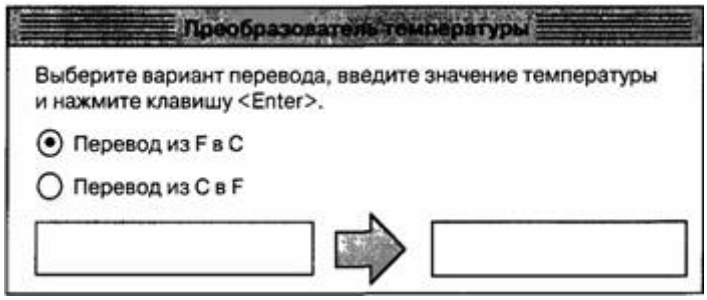

Рис. 7.1. Вариант диалогового окна с использованием группы переключателей

- Перемещение руки к графическому устройству ввода данных:
  H
- Перемещение курсора к необходимому переключателю в группе:
  H P
- Нажатие на необходимый переключатель:
  H P K

В половине случаев в интерфейсе уже будет выбрано требуемое направление перевода, и поэтому лаборанту не придется кликать на переключатель. Сейчас мы рассматриваем случай, когда переключатель не установлен в требуемое положение.

- Перемещение рук снова к клавиатуре:

H P K H

- Ввод четырех символов:
  H P K H K K K K
- Нажатие клавиши <Enter>:
  H P K H K K K K K

Нажатие клавиши <Enter> завершает часть анализа, касающуюся метода. В соответствии с правилом 0 мы ставим оператор M перед всеми операторами K и P за исключением операторов P, указывающих на аргументы, которых в нижеследующем примере нет:

H M P M K H M K M K M K M K M K

Правило 1 предписывает заменить P M K на P K и удалить все другие операторы M, которые являются ожидаемыми (в указанном примере таких нет). Кроме того, правило 2 предписывает удалять операторы M в середине цепочек. После применения этих двух правил остается следующая запись:

H M P K H M K K K K M K

В соответствии с правилом 4 следует оставить оператор M перед конечным K. Правила 3 и 5 в данном примере не применяются.

Следующий шаг — это заменить символы операторов на соответствующие временные интервалы (напомним, что K=0.2; P=1.1; H=0.4; M=1.35).

H + M + P + K + H + M + K + K + K + K + M + K = 0.4 + 1.35 + 1.1 + 0.2 + 0.4 + 1.35 + 4*(0.2) + 1.35 + 0.2 = 7.15 с

В случае когда переключатель уже установлен в требуемое положение, метод действий становится следующим:

M K K K K M K

M + K + K + K + K + M + K = 3.7 с

По условиям задачи оба случая являются равновероятными. Таким образом, среднее время, которое потребуется лаборанту на использование интерфейса для перевода из одной шкалы в другую, составит (7.15+3.7)/2 ≈ 5.4 с. Но поскольку описанные два метода являются разными, лаборанту

будет трудно использовать их автоматично. Нерешенной проблемой количественных методов анализа остается оценка процента появления ошибок при использовании данной модели интерфейса.

Далее мы рассмотрим графический интерфейс, в котором используется известная всем метафора.

7.5.2. Интерфейс для программы: вариант 2. Графический интерфейс пользователя

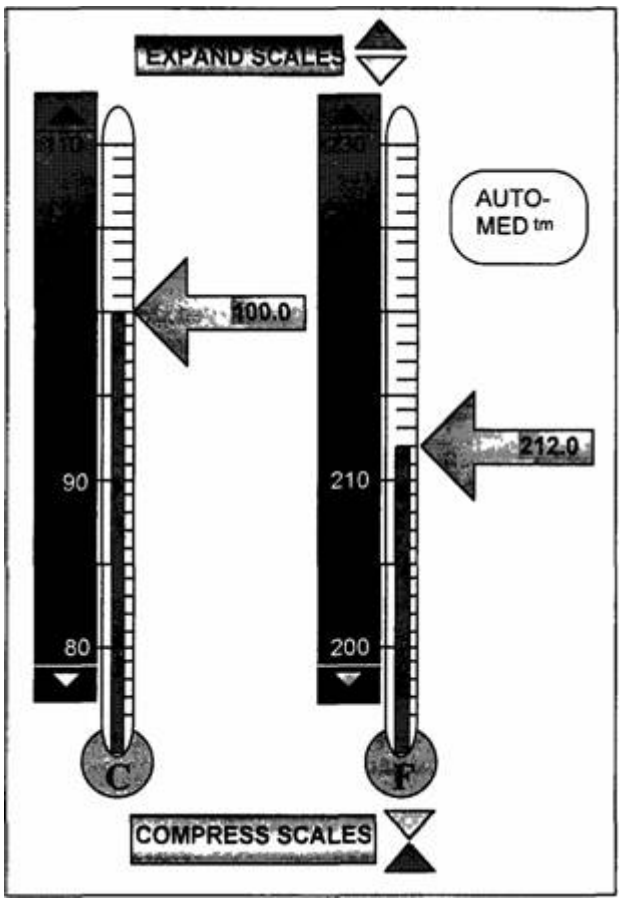

Рис. 7.2. Графический интерфейс пользователя

В интерфейсе, показанном на рис. 7.2, используется наглядное отображение термометров. Лаборант может поднять или опустить указатель на каждом термометре методом перетаскивания с помощью ГУВ. Лаборант определяет, какой ему необходимо сделать пересчет, перемещая стрелку указателя либо по шкале Цельсия, либо по шкале Фаренгейта. Лаборанту не требуется вводить символы посредством клавиатуры — он просто выбирает значение температуры на одном из термометров. При перемещении указателя

на одном термометре указатель на другом перемещается на соответствующее значение. Точность устанавливается с помощью регуляторов масштабирования шкал. Также возможно изменить текущий диапазон значений. Изменение шкалы или диапазона на одном термометре автоматически приводит к соответствующему изменению на другом. Точное числовое значение отображается на перемещаемой стрелке. Температура показывается как в числовом виде так и с помощью уровня градусника, поэтому лаборант может, на свое усмотрение, пользоваться либо графическим вариантом представления данных, либо символьным. Сервис «Автомед» позволяет установить диапазоны термометров с центром в районе 37 градусов шкалы Цельсия и 98.6 градусов Фаренгейта на случай, если кто-то из сотрудников работает со значениями температуры тела человека. Эта опция служит для экономии времени.

С помощью нажатия кнопок «Расширить шкалу» (Expand Scales) и «Сжать шкалу» (Compress Scales) можно уменьшить или увеличить цену деления шкал в 10 раз. Для перехода к значению, которое в данный момент не видно на экране, лаборант расширяет шкалу, затем прокручивает до нужного места на шкале, устанавливает стрелку на необходимое температурное значение и потом сжимает шкалу до получения требуемой точности, при необходимости подстраивая стрелку указателя.

Провести анализ этого графического интерфейса с помощью модели скорости печати GOMS довольно сложно, поскольку способ, которым лаборант может его использовать, зависит от того, где в данный момент установлена стрелка указателя, какой необходим диапазон температур и какая требуется точность. Рассмотрим сначала простой случай, при котором диапазон температурных шкал и точность перевода уже находятся в желаемом положении. Анализ позволит определить минимальное время, необходимое для использования этого интерфейса.

- Запишем, какие жесты использует лаборант, когда перемещает руку к ГУВ, щелкает по кнопке и удерживает ее, указывая на стрелку одного из термометров:

  H P K

- Продолжим записывать те жесты, которые использует лаборант для перемещения стрелки к необходимому температурному значению и отпускает кнопку ГУВ

  H P K P K

- Поставим операторы M в соответствии с правилом 0:

  H M P M K M K

- Удалим два оператора M в соответствии с правилом 1:

  H M P K K

Когнитивные единицы, разделители последовательностей и т.д. здесь не используются, поэтому правила 2-5 не применяем. Складывая значения операторов, получаем общее время:

H M P K K

0.4 + 1.35 + 1.1 + 0.2 + 0.2 = 3.25 с

Результат вычисления относится к удачному случаю, когда исходный термометр уже предустановлен на требуемый диапазон и точность. Теперь рассмотрим случай, при котором лаборант расширяет шкалу, чтобы увидеть необходимое температурное значение, изменяет диапазон, сжимает шкалу, чтобы получить требуемую точность, и затем перемещает стрелку указателя. Далее приводится общая запись метода, который использует лаборант, без промежуточных шагов. (Мы исходим из того, что лаборант является опытным пользователям и не прокручивает шкалу туда и обратно, чтобы найти на ней нужный участок.) Лаборанту приходится несколько раз пользоваться стрелками для прокрутки температурной шкалы. На каждую операцию прокручивания экрана может потребовать нескольких жестов. Кроме того, требуется время на то, чтобы отобразить изменения на экране, связанные с его прокруткой. Чтобы оценить время прокручивания, был построен такой интерфейс и измерены эти значения. Все они были равны 3 с

и более. Обозначая время прокручивания шкал через S, запишем последовательность жестов, которые применяет лаборант.

H P K S K P K S K P K S K P K K

В соответствии с правилами расставляем операторы M:

H + 3(M + P + K + S + K) + M + P + K + K

0.4 + 3*(1.35 + 0.2 + 3.0 + 0.2) + 1.35 + 0.4 + 0.2 + 0.2 = 16.8 с

За исключением редких случаев, когда шкалы уже с самого начала установлены правильно, идеальному пользователю понадобится более 16 с на то, чтобы выполнить перевод из одной шкалы в другую, тогда как реальный, т.е. не идеальный пользователь, может сбивать шкалы и стрелки указателей, и поэтому ему понадобится даже больше времени.

7.6. Измерение эффективности интерфейса

Мы рассмотрели два интерфейса: в одном из которых требуется около 5 с на выполнение задачи, а в другом — более 15 с. Отсюда ясно, какой из интерфейсов лучше удовлетворяет поставленным условиям. Следующий вопрос — это определить, насколько быстро работает тот интерфейс, который отвечает поставленным требованиям.

Если имеется модель интерфейса, то с помощью GOMS и его расширений можно определить время, необходимое пользователю на выполнение любой, четко сформулированной задачи, для которой данный интерфейс предусмотрен. Однако модели анализа не могут дать ответ на вопрос о том, насколько быстро должен работать интерфейс. Чтобы ответить на него, мы можем воспользоваться мерой, применяемой в теории информации. Далее мы будем рассматривать термин *информация* в техническом смысле, т.е. как квантификацию некоторого объема данных, передаваемых с помощью средства коммуникации, как, например, при разговоре двух людей по телефону, или если человек подает некоторый сигнал машине, например с помощью нажатия кнопки ГУВ, когда курсор находится в определенной области экрана. Перед тем как углубиться в детали

техники измерения того, какой объем информации нужен для выполнения поставленной задачи, обоснуем необходимость такого измерения.

Чтобы сделать правильную оценку времени, необходимого на выполнение задачи с помощью самого быстрого интерфейса, прежде всего следует определить минимальное количество информации, которое пользователь должен ввести, чтобы выполнить задачу. Это минимальное количество не зависит от модели интерфейса. Если методы работы, используемые в предполагаемом интерфейсе, требуют введения такого количества информации, которое превышает минимальное, это означает, что пользователь делает лишнюю работу, и поэтому интерфейс можно усовершенствовать. С другой стороны, если от пользователя требуется ввести именно то количество информации, которое необходимо для выполнения задачи, то для этой задачи интерфейс нельзя сделать более производительным путем изменения количества информации. В этом случае пути улучшения интерфейса (а также много путей для ухудшения) все же остаются, но по крайней мере данная цель повышения производительности будет уже достигнута.

Информационно-теоретическая производительность определяется так же, как понятие производительности определяется в термодинамике — отношением мощности на выходе к мощности на входе процесса. Если в течение какого-то периода времени электрогенератор, работающий от двигателя производительностью в 1000 ватт, производит 820 ватт, то он имеет производительность 820/100=0.82. Производительность также часто обозначается через проценты. В этом случае производительность электрогенератора будет составлять 82%. Идеальный генератор (который не может существовать с точки зрения второго закона термодинамики) должен иметь производительность 100%.

Информационная производительность интерфейса *E* определяется как отношение минимального количества информации, необходимого для выполнения задачи, к количеству информации, которое должен ввести

пользователь. Так же как и в отношении физической производительности, параметр E может изменяться в пределах от 0 до 1. Если никакой работы для выполнения задачи не требуется или работа просто не производится, то производительность составляет 1. (Это формальное положение вводится для того, чтобы избежать деления на 0, как в случае ответа на выводимое прозрачное сообщение об ошибке.)

Производительность E может равняться и 0 в случаях, когда пользователь должен ввести информацию, которая совершенно бесполезна. Следует отметить, что в интерфейсах можно встретить немало деталей, которые имеют сомнительную ценность из-за параметра $E=0$. Примером такого бесполезного элемента может быть диалоговое окно, в котором есть только одна-единственная возможность для действия пользователя, например кнопка OK. (В JavaScript есть даже специальная команда Alert, предназначенная только для того, чтобы делать такие ненужные диалоговые окна. Разработчики языка JavaScript были достаточно разумны, чтобы убрать из него команду goto и сделать программирование на этом языке структурным, но они упустили из виду аспект интерфейса.)

В параметре E учитывается только информация, необходимая для задачи, и информация, вводимая пользователем. Два или более методов действия могут иметь одинаковую производительность E, но иметь разное время выполнения. Возможно даже, что один метод имеет более высокий показатель E, но действует медленнее, чем другой метод, — например $M\ K\ M\ K$ и $M\ K\ K\ K$. В этом примере при использовании первого метода должно быть введено только два символа. При использовании второго метода требуется ввести три символа, но времени на все действие тратится меньше. Трудно привести другие примеры из обычной жизни, в которых происходит аналогичная перестановка скорости и информационной производительности. Как правило, чем более производительным является интерфейс, тем более продуктивным и более человекоориентированным он является.

Информация измеряется в битах. Один бит, который представляет собой один из двух альтернативных вариантов (таких как 0 или 1, да или нет), является единицей информации. Например, чтобы выбрать один из каких-либо четырех объектов, потребуется 2 бита информации. Если объекты обозначить как A, B, C и D, первый бит информации определит выбор между A и B или C и D. Когда первый выбор сделан (например, C и D), второй бит определит выбор между следующими двумя элементами (либо C, либо D). Двух двоичных выборов, или двух битов, достаточно для выбора одного элемента из четырех. Чтобы сделать выбор из группы восьми элементов, потребуется 3 бита. Из шестнадцати элементов — 4 бита, и т.д. В общем случае при количестве n равновероятных вариантов суммарное количество передаваемой информации определяется как степень 2, равная n:

$\log_2 n$

Количество информации для каждого варианта определяется как

$$(1/n) \log_2 n \qquad (1)$$

Если вероятности для каждой альтернативы не являются равными и i-я альтернатива имеет вероятность $p(i)$, то информация, передаваемая этой альтернативой, определяется как

$$p(i) \log_2 (1/p(i)) \qquad (2)$$

Количество информации является суммой (по всем вариантам) выражения (2), которое при равновероятных вариантах сводится к выражению (1). Отсюда следует, что информационное содержание интерфейса, в котором возможно сделать только нажатие единственной клавиши (а ненажатие клавиши не допускается), составляет 0 бит:

$$1 \log_2 (1) = 0 \qquad (3)$$

Однако может показаться, что нажатие единственной клавиши способно, например, вызвать подрыв динамита для разрушения здания. Таким образом, передает ли это нажатие какую-нибудь информацию? На самом деле нет, потому что ненажатие кнопки не было предусмотрено как альтернатива —

интерфейс допускает «только нажатие единственной клавиши». Если же нажатие клавиши не производится в течение 5-минутного периода, когда подрыв возможен, то здание не будет разрушено, и поэтому нажатие или ненажатие передает до 1 бита информации, так как в этом случае имеется альтернатива из двух вариантов. Из выражения (2) следует, что в вычислениях используется вероятность (p) того, что здание будет разрушено. Таким образом, вероятность того, что оно не будет разрушено, составляет $1-p$. С помощью выражения (2) мы можем вычислить информационное содержание данного интерфейса:

$$p \log_2 (1/p) + (1-p) \log_2 (1/(1-p)) \qquad (4)$$

При $p=S$ результат выражения (4) составит:

$$S*1 + S*1 = S + S = 1$$

Значение выражения (4) будет меньше 1, если $p \neq S$. В частности при $p = 0$ или $p = 1$ оно составит 0, как это видно из выражения (3).

Этот пример показывает важный момент, который заключается в том, что мы можем оценить объем информации, содержащейся в сообщении, только в контексте всего набора возможных сообщений. Чтобы подсчитать количество информации, передаваемой некоторым полученным сообщением, необходимо знать в частности вероятность, с которой это сообщение может быть отправлено. Количество информации в любом сообщении не зависит от других сообщений, которые были в прошлом или могут быть в будущем, не связано со временем или продолжительностью и не зависит от каких-либо иных событий, так же как результат подбрасывания симметричной монеты не зависит от результата предыдущих подбрасываний или от времени дня, когда это подбрасывание производится.

Кроме того, важно учитывать, что:

«нельзя путать понятие информации с понятием смысла ...информация является мерой свободы выбора сообщения... Следует отметить, что при наличии только двух возможных сообщений утверждать, что какое-то сообщение передает какой-то объем (1 бит) информации, неправильно.

Понятие информации не применимо к отдельным сообщениям (в отличие от понятия смыла), но применимо к ситуации в целом; при этом единица информации показывает, что в данной ситуации имеется некоторый объем свободы в выборе сообщения, который удобно обозначать как стандартный или единичный объем информации».

Однако действия, которые совершает пользователь при выполнении задачи, можно с большей точностью смоделировать в виде процесса Маркова, в котором вероятность последующих действий зависит от уже совершенных пользователем действий. Тем не менее, для данного рассмотрения достаточно использовать упомянутые вероятности отдельных, единичных событий, при этом будем исходить из того, что все сообщения являются независимыми друг от друга и равновероятными.

Также можно вычислить количество информации, которое передается с помощью устройств, отличающихся от клавиатуры. Если экран дисплея разделен на две области — со словом «Да» в одной области и словом «Нет» — в другой, то один клик, совершенный в одной из областей, будет передавать 1 бит информации. Если имеется n равновероятных объектов, то нажатием на один из них сообщается $\log_2 n$ бит информации. Если объекты имеют разные размеры, то количество информации, сообщаемой каждым из них, не изменяется, но увеличивается время перемещения ГУВ к более мелким объектам (далее мы покажем способ вычисления этого времени). Если объекты имеют разные вероятности, формула остается аналогичной той, которая была дана для случая ввода с клавиатуры разновероятных данных. Различие состоит только в том, что для нажатия клавиши может потребоваться 0.2 с. тогда как для нажатия кнопки, изображенной на экране, в среднем может потребоваться около 1.3 с (без учета времени перемещения руки пользователя с клавиатуры на ГУВ).

В случае голосового ввода информации его информационное содержание можно вычислить, если рассматривать речь как

последовательность вводимых символов, а не как непрерывный поток определенного диапазона и продолжительности.

Данный подход к теории информации и ее связи с разработкой интерфейсов является упрощенным. Но даже в такой упрощенной форме, которую мы также использовали при рассмотрении модели GOMS, теория информации может дать нам общий критерий оценки качества интерфейса.

7.6.1. Производительность интерфейса

Полезно подробно рассмотреть пример вычисления среднего количества информации, требуемого для некоего интерфейса. Для этого мы снова используем пример интерфейса для перевода температур из одной шкалы в другую. В соответствии с условиями требуется, чтобы количество символов, вводимых в температурный преобразователь, равнялось в среднем 4. Кроме того, по условиям задачи десятичная точка используется однократно в 90% вводимых данных, а в 10% — вообще не встречается; знак минус появляется один раз в 25% данных и совсем не встречается в остальных 75% данных. Из соображений простоты, а также потому, что не требуется ответ с точностью до 1%, мы будем исходить из того, что все остальные цифры встречаются с одинаковой частотой, и не буду учитывать те 10% данных, которые не содержат десятичной точки.

Требуется определить множество возможных вариантов ввода и их вероятности. Возможны 5 вариантов (d означает цифру):

1. -.dd
2. -d.d
3. .ddd
4. d.dd
5. dd.d

Первые два варианта встречаются в 12.5% случаев, и количество каждого из них составляет 100. Каждый из последних трех вариантов встречается в 25% случаев, и количество каждого из них составляет почти

1000. Вероятность каждого из первых двух вариантов ввода составляет (0.125/100)=0.00125. Вероятность любого из последних трех вариантов ввода составляет (0.75/3000)=0.00025. Сумма вероятностей, как это и должно быть, составляет 1.

Количество информации (в битах), передаваемое каждым вариантом, определяется выражением (2):

$p(i) \log_2 (1/p(i))$

Значение этого выражения составляет приблизительно 0.012 для отрицательных значений ввода и 0.003 — для положительных. 200*0.00674+3000*0.003 дает в сумме 11.4 бита для каждого варианта ввода.

Важно учесть вероятности вариантов. Если использовать простой подход, в котором все 12 символов (минус, десятичная точка и 10 цифр) принять как равновероятные, то вероятность каждого символа составит 1/12, а количество информации, содержащейся в 4-значном варианте ввода, составит приблизительно

$4 \log_2(12) \approx 14$ бит

В теории информации есть теорема, в которой утверждается, что максимум информации передается при условии, что все символы равновероятны. Поэтому если принять все варианты как равновероятные, то общее значение будет равно количеству информации в каждом отдельном варианте или превышать его. Очевидно, что такое допущение позволяет упростить вычисление информационного содержания. Если же результирующее значение приближенного вычисления меньше количества информации, которое пользователь должен ввести в интерфейс, то проводить еще более точные вычисления уже нет необходимости.

Мы только что выяснили, что каждый раз, когда лаборанту требуется провести преобразование температурных значений, он должен ввести в среднем около 11 бит информации. Мы можем разделить это количество на то количество информации, которое требуется ввести в интерфейс, что мы

сейчас и сделаем. В результате мы получим производительность (эффективность) данного интерфейса.

Другим упрощением, позволяющим провести быстрый анализ интерфейса, является вычисление различных жестов на основе количества информации, передаваемого одним нажатием клавиши или одной операцией ГУВ. При передаче информации нажатием клавиши ее количество зависит от общего количества клавиш и относительной частоты использования каждой из них. Таким образом, нажатия клавиши могут использоваться как приблизительная мера информации. Если на клавиатуре имеется 128 клавиш, и каждая из них используется с одинаковой частотой, то нажатие любой из них будет передавать 7 бит информации. В действительности частота использования клавиш субщественно различается. Например, пробел или буква е используются чаще, чем й или \, поэтому в большинстве приложений на каждое нажатие клавиши приходится в среднем около 5 битов. По условиям нашей задачи среднее число символов вводимых температурных значений не должно превышать 4.

Для данного анализа удобнее использовать более простую меру, чем теоретическая информационная производительность. **Символьная эффективность** часто имеет такую же практическую ценность, что и информационная производительность. Она определяется как минимальное количество символов, необходимое для выполнения задачи, отнесенное к количеству символов, которое в данном интерфейсе требуется ввести пользователем.

Если в нашем интерфейсе потребуется вводить в среднем 4 символа, то символьная эффективность такого интерфейса составит 100%. При добавлении еще одной клавиши, обозначающей шкалу перевода температурного значения, а также еще одной для разделения, средняя длина ввода возрастет до 6 символов, а символьная эффективность снизится до 67%. Если в качестве устройства ввода лаборант будет использовать числовую клавиатуру, состоящую из 16 клавиш, то каждой отдельной

клавишей будет передаваться 4 бита информации, и поэтому производительность интерфейса возрастет. (Однако по условиям задачи такой возможности не предусмотрено.)

Поскольку любая задача в соответствии с анализом GOMS требует как минимум одного ментального оператора, наиболее производительный интерфейс с использованием клавиатуры для перевода температурных значений из одной шкалы в другую будет теоретически иметь следующее среднее время использования:

M + K + K + K + K = 2.15 с

Таким образом, он будет значительно быстрее, чем любой из двух уже рассмотренных вариантов. Однако введение 4 символов с помощью стандартной клавиатуры дает, по крайней мере, 20 бит информации, тогда как требуется только 10. Следовательно, теоретическая информационная производительность составляет 55%, а значит, существует возможность улучшения. Как мы уже видели, использование стандартной числовой клавиатуры вместо полной клавиатуры снижает объем информации, вводимой на каждые 4 символа, до 16 бит, что повышает производительность до 60%. Желаемая числовая клавиатура, содержащая только цифры, знак минус и десятичную точку, позволит немного повысить производительность — до 70%. Дальнейшее повышение производительности возможно через использование особых кодировок обозначений температуры и изобретение новых устройств ввода, но здесь возникают трудности, связанные с обучением и лишними расходами, поэтому остановимся на варианте с 70% теоретической информационной производительности. Независимо от того, могут ли теоретические границы быть достигнуты на практике или нет, они дают нам направление в разработке интерфейса.

7.6.2. Другие решения интерфейса

В разделе 7.6.1 мы приостановили дальнейшие попытки улучшения интерфейса, достигнув 70% теоретической информационной

производительности. Данная производительность определена для пока еще неизвестного, теоретического интерфейса, в котором каким-то образом можно получить 100% эффективность использования клавиш. Давайте посмотрим, насколько мы можем приблизиться к этому идеалу с помощью стандартной клавиатуры и ГУВ.

Рассмотрим интерфейс, в котором используется клавиатура со всеми символами. В таком интерфейсе на экране появляется следующее сообщение:

> Для перевода температуры из одной шкалы в другую укажите нужную шкалу с помощью символа C (шкала Цельсия) или F (шкала Фаренгейта). Введите числовое значение температуры, затем нажмите клавишу Enter. Результат преобразования будет отображен на экране.

GOMS-анализ показывает, что пользователь должен сделать 6 нажатий клавиш. По правилам расстановки операторов M получаем .следующую запись:

M K K K K K M K

Среднее время составит 3.9 с.

Мы можем уменьшить это время, если сами символы C и F будем использовать в качестве разделителей. Рассмотрим интерфейс, в котором появляется следующая инструкция:

> Для перевода температуры из одной шкалы в другую введите числовое значение температуры и следом поставьте символ C, если оно в шкале Цельсия, или F, если оно в шкале Фаренгейта. Результат преобразования будет отображен на экране.

В данном примере нажимать на клавишу <Enter> не требуется. Некоторые примитивные средства разработки интерфейсов требуют, чтобы пользователь обязательно использовал клавишу <Enter>, и поэтому в них невозможно использовать символы C и F в качестве разделителей. Такие

инструменты не подходят для разработки человекоориентированных интерфейсов.

GOMS-анализ показывает, что для интерфейса с символами C и F в качестве разделителей запись будет следующей:

M K K K K M K

Среднее время составит 3.7 с. Если бы мы не знали, что теоретически минимальное время составляет 2.15 с, то это решение могло бы показаться удачным. Оно является значительно более эффективным, чем ранее рассмотренные, поэтому мы могли бы на нем остановиться. Однако теоретический минимум подстегивает нас к поиску еще более быстрой интерфейсной модели. Рассмотрим интерфейс, изображенный на рис. 7.3. Такой интерфейс можно назвать *разветвленным*. В нем один ввод дает в результате два вывода.

Рис. 7.3. Интерфейс, в котором не используется разделитель. Более эффективным является вариант, в котором выполняется посимвольная обработка вводимых данных и одновременное преобразование в обеих шкалах

В разветвленном интерфейсе нет необходимости в разделителе. Кроме того, пользователю не нужно указывать, какое именно преобразование требуется провести. GOMS-анализ показывает, что для 4 символов, которые в среднем будут вводиться, запись будет следующей:

M K K K K

В разветвленном интерфейсе достигается минимальное время 2.15 с, и его символьная эффективность составляет 100%.

Если, как в нашем примере, в месте вывода происходит изменение результата в тот момент, когда символы вводятся, это колебание цифр не отвлекает пользователя, потому что в локусе его внимания находится именно ввод данных. Непрерывно изменяемые значения на выводе могут быть даже полезными — после нескольких применений пользователь будет замечать эти колебания только краем глаза, что будет служить подсказкой о том, что система отвечает на вводимые данные. Если используется посимвольный режим работы интерфейса, то для большей эффективности такая система должна реагировать довольно быстро. В частности скорость реакции не должна быть меньше скорости ввода. Такая проблема может возникать только при сетевом использовании интерфейса.

Хотя это и не входит в условие задачи, но может возникнуть вопрос о том, как происходит очистка полей в преобразователе для выполнения следующего преобразования. Добавляет ли операция очистки еще одно нажатие клавиши? Необязательно. Например, мы можем разработать интерфейс таким образом, что каждый раз, когда оператор возвращается к своему основному занятию или переходит к другой задаче, значения, указанные в преобразователе, могут автоматически затеняться, а сам температурный преобразователь становиться неактивным; причем в это время все значения могут оставаться в своих полях так, что при необходимости их можно было бы опять увидеть, но уже при следующем вводе все они будут стерты.

Разветвленный интерфейс не обязательно является самым лучшим из тех, что уже были рассмотрены — или из тех, что возможны, — только потому, что он имеет оптимальную скорость работы и является весьма эффективным. Кроме скорости есть и другие параметры, которые также являются важными: частота появления ошибок; время, необходимое для изучения интерфейса; возможность длительного запоминания способа использования интерфейса. Особенно следует обратить внимание на частоту появления ошибок в разветвленном интерфейсе, поскольку есть вероятность

того, что лаборант может прочитать результат не из того поля. Это важно особенно потому, что когда он услышал, например слово *Цельсий*, ему необходимо прочитать значение из поля шкалы Фаренгейта. Тем не менее, разветвленный температурный преобразователь определенно входит в небольшое число тех интерфейсов, которые можно рассматривать как рабочие варианты для программы преобразования температурных значений из одной шкалы в другую. Другие рассмотренные нами примеры, которые могли бы показаться удачными, если бы мы не проводили с ними GOMS-анализ, на самом деле не являются таковыми.

Используемая как в простом анализе временных затрат на нажатия клавиш, так и в полном информационно-теоретическом исследовании, квантификация теоретического интерфейса с минимальным временем использования или с минимальным количеством используемых символов, или с минимальным количеством используемой информации может быть полезна с точки зрения разработки интерфейса. *Без количественных ориентиров мы можем только догадываться о том, насколько хорошо мы разработали интерфейс и есть ли возможность его улучшения.*

7.7. Закон Фитса и закон Хика

Различные количественные законы, имеющие отношение к разработке интерфейсов, имеют хорошее когнитивное обоснование, и их правильность была неоднократно проверена. Эти законы часто дают нам дополнительные данные, на основе которых можно принимать те или иные решения, связанные с разработкой интерфейсов. Закон Фитса (Fitts' Law) позволяет определить количественно тот факт, что чем дальше находится объект от текущей позиции курсора или чем меньше размеры этого объекта, тем больше времени потребуется пользователю для перемещения к нему курсора. Закон Хика (Hick's Law) позволяет количественно определить наблюдение, заключающееся в том, что чем больше количество вариантов заданного типа вы предоставляете, тем больше времени требуется на выбор.

7.7.1. Закон Фитса

Представим, что вы перемещаете курсор к кнопке, изображенной на экране. Кнопка является *целью* данного перемещения. Длина прямой линии, соединяющей начальную позицию курсора и ближайшую точку целевого объекта, определяется в законе Фитса как *дистанция*. На основе данных о размерах объекта и дистанции **закон Фитса** позволяет найти среднее время, за которое пользователь сможет переместить курсор к кнопке.

В одномерном случае, при котором размер объекта вдоль линии перемещения курсора обозначается как *S*, а дистанция от начальной позиции курсора до объекта — как *D* (рис. 7.4), закон Фитса формулируется следующим образом:

$$Время\ (мс) = a + b \log_2 (D/S + 1)$$

(Константы a и b устанавливаются опытным путем по параметрам производительности человека.)

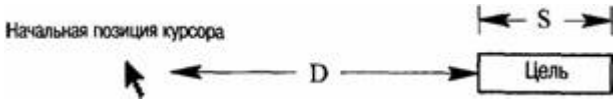

Рис. 7.4. Расстояния, которые используются в законе Фитса для определения времени, необходимого для перемещения курсора к цели

Вычисляемое время отсчитывается от момента, когда курсор начинает движение по прямой линии, до момента, когда пользователь щелкает мышью по целевому объекту. Логарифм по основанию 2 является мерой трудности задачи в количестве бит информации, которое требуется для описания (одномерного) пути перемещения курсора.

Для вычисления времени можно использовать любые единицы измерения дистанции, т.к. *D/S* является отношением двух дистанций и поэтому не зависит от единицы измерения. Отсюда следует, что хотя указательное устройство может переместиться на расстояние большее или меньшее, чем то расстояние, на которое переместится на экране курсор, закон все равно работает, при условии, что соотношение между движением ГУВ и курсора является линейным. Закон Фитса может применяться только к

тем типам перемещения, которые совершаются при использовании большинства человеко-машинных интерфейсов, т.е. к таким перемещениям, которые невелики относительно размеров человеческого тела и которые являются непрерывными (совершаемыми одним движением). Для приближенных вычислений мы используем следующие значения констант в уравнении закона Фитса: $a=50, b=150$.

7.7.2. Закон Хика

Перед тем как переместить курсор к цели или совершить любое другое действие из набора множества вариантов, пользователь должен выбрать этот объект или действие. В законе Хика утверждается, что когда необходимо сделать выбор из n вариантов, время на выбор одного из них будет пропорционально логарифму по основанию 2 от числа вариантов плюс 1, при условии, что все варианты являются равновероятными. В этом виде закон Хика очень похож на закон Фитса:

$$Время\ (мс) = a + b \log_2 (n+1)$$

Если вероятность 1-го варианта равна $p(i)$, то вместо логарифмического коэффициента используется

$$S_i\ p(i) \log_2 (1/p(i)+1)$$

Коэффициенты, используемые в выражении закона Хика, в большой степени зависят от многих условий, включая то, как представлены возможные варианты, и то, насколько хорошо пользователь знаком с системой. (Если варианты представлены непонятным образом, значения a и b возрастают. Наличие навыков и привычек в использовании системы снижает значение b.) Мы не будем рассматривать эти зависимости — для нас важно, что для принятия того или иного решения требуется время; что для принятия сложных решений требуется больше времени, чем для принятия простых решений; и что взаимосвязь является логарифмической. При отсутствии более точных данных для проведения быстрых и приблизительных

вычислений мы можем воспользоваться теми же значениями a и b, которые использовали для закона Фитса.

При использовании любых положительных и ненулевых значений a и b из закона Хика следует, что предоставление пользователю сразу нескольких вариантов одновременно обычно является более эффективным, чем организация тех же вариантов в иерархические группы. Выбор из одного меню, состоящего из 8 элементов, производится быстрее, чем из двух меню, состоящих их 4 элементов каждое. Если все элементы могут быть выбраны с равной вероятностью и если не учитывать время, необходимое для открытия второго меню (которое, конечно, еще более увеличило бы время для интерфейса, состоящего из двух меню), то сравнение времени для выбора одного элемента из восьми ($a + b \log_2 8$) с удвоенным временем для выбора одного элемента из четырех 2 ($a + b \log_2 4$) покажет, что

$$a + 3b < 2(a + 2b)$$

поскольку $\log_2 8 = 3$, а $\log_2 4 = 2$, а также поскольку $a<2a$ и $3b<4b$.

Это согласуется с данными, полученными в экспериментах со структурами меню.