Санкт-Петербургский государственный университет

Математико-механический факультет

Лаборатория СПРИНТ

Введение в разработку приложений на платформе Atom/MeeGo для нетбуков и планшетников

О. Н. Граничин, А. В. Корявко, М. А. Антропова

Учебное пособие

Санкт-Петербург
2011

ББК __.__
Г __

Р е ц е н з е н т ы: канд. физ.-мат. наук, доцент В. И. Кияев
(С.-Петерб. гос. ун-т)
канд. физ.-мат. наук, доцент И. О. Одинцов
(менеджер по стратегическому развитию Intel)

Печатается по решению Редакционно-издательского совета математико-механического факультета Санкт-Петербургского государственного университета

Г р а н и ч и н О.Н., К о р я в к о А.В., А н т р о п о в а М.А.
Г __ Введение в разработку приложений на платформе Atom/MeeGo для нетбуков и планшетников. Учебное пособие. – СПб., 2011. – 139с.

ISBN

В учебном пособии представлены материалы одноименного курса по разработке приложений для мобильных устройств, прочитанного авторами для слушателей Летней школы 2011 г. Лаборатории системного программирования и информационных технологий (СПРИНТ, СПбГУ-Intel). Рассматриваются как общие основы разработки приложений для мобильных устройств, так и специальные инструменты для программирования нетбуков и планшетников, ориентированные на новую операционную систему MeeGo и процессоры Intel Atom. Особое внимание уделено новым характеристикам разработки программного обеспечения: User eXxpirience/Usability, а также разработка «сквозных» приложений и интерфейсов для разных устройств Computing Continuum.

Пособие предназначено для студентов, аспирантов и преподавателей соответствующих специальностей, а также может быть полезно широкому кругу читателей.

Библиография – 49 назв., таблиц – 1, рисунков – 5.

ББК __.__

ISBN

Содержание

Введение

Назначение, структура и авторы курса.

Основу пособия составил курс лекций по разработке приложений для нетбуков и планшетников, прочитанный авторами летом 2011 года для слушателей Летней школы Лаборатория СПРИНТ (Системного ПРограммирования и ИНформационных Технологий) СПбГУ, созданной и финансируемой при поддержке корпорации Интел.

Главная цель пособия — дать общее представление о процессе разработки приложений для нетбуков и планшетников на платформе Atom/MeeGo. Настоящее пособие следует рассматривать в качестве продолжения общего вводного курса «Введение в разработку приложений на платформе Atom/MeeGo», разработанного ранее О.Н. Граничиным, В.И. Кияевым, А.В. Корявко, С.А. Леви, К.С. Амелиным, Е.И. Антал и В.И. Васильевым.

Курс состоит из введения, пяти лекций, пяти лабораторных работ и заключения. Первая лекция носит общий характер: перспективы развития средств вычислительной техники компьютерный континуум Intel, сведения о процессорах Intel-Atom и операционной системе MeeGo. Вторая – об общих средствах и приемах разработки мобильных приложений в Linux. Третья – о специализированных инструментах разработки для MeeGo, фреймворке Qt и магазине приложений Intel AppUp. В четвертой лекции показывается как разрабатывать приложения для планшетников. Пятая – посвящена особенностям психологии человеко-компьютерного взаимодействия. В лабораторных работах на работоспособных изолированных примерах приложений иллюстрируется лекционный материал. Лекция 1 составлена О.Н. Граничиным, 2,3 – доработаны А.В. Корявко на основе соответствующих лекций С.А. Леви из предшествующего курса, 4 – составлена А.В. Корявко, 5 – М.А. Антроповой, лабораторные работы 1-5 составлены А.В. Корявко. Учебные материалы доступны по ссылке http://www.math.spbu.ru/user/gran/AtomMG2.

Авторы благодарят сотрудников корпорации Интел Алексея Владимировича Николаева и Игоря Олеговича Одинцова за инициативу по разработке и созданию курса, за активное участие в разработке программы курса.

При подготовке пособия были использованы материалы, размещенные на сайтах: www.meego.com, www.intel.com, www.nokia.com и др.

Ссылки на программные продукты различных российских и зарубежных фирм не используются в рекламных целях и носят исключительно иллюстративный или справочный характер.

Пособие издано при финансовой поддержке Лаборатории СПРИНТ СПбГУ.

1. Компьютерный континуум, процессор Intel-Atom. ОС MeeGo

Тенденции развития вычислительной техники – к системам на кристалле. Компьютерный континуум Intel и процессор Intel-Atom. ОС MeGoo. Средний слой ОС (Middleware) .

1.1. Тенденции развития вычислительной техники – к системам на кристалле

Взглянем на историю развития средств вычислительной техники. За шесть десятилетий пройден путь от ламп, через транзисторы, интегральные микросхемы к сверхбольшим интегральным микросхемам. Что будет дальше? Основной вопрос – как обрабатывать данные? Для людей старшего поколения знакомство с компьютерами начиналось с фантастических романов. Еще школьниками, не видя никогда настоящих компьютеров, многие из них были уверены, что скоро появится «Искусственный Разум», освободив нас от многих рутинных забот. Наверное поэтому люди старшего поколения, занятые в развитии информационных технологий (ИТ), острее чувствуют тенденции кардинальных преобразований. Конечно, кто-то возразит, что многие не дождавшись за 60 лет «искусственного разума», разочаровались в перспективах, тем более, что по мере развития ИТ кроме такой фантастической цели появилось огромное количество простых и сложных конкретных задач.

Новые потребности, глобализация задач, экспоненциальное возрастание сложности вычислительных систем и наметившаяся в последнее время тенденция по преодолению отставания отечественной ИТ отрасли в развитии суперкомпьютерных вычислений (Т-Платформы, СКИФ-Аврора и др. проекты) заставляют уже в практическом плане задуматься о перспективах и возможной смене парадигмы: «Что такое процесс вычислений?» Объективные сегодняшние тенденции – миниатюризация и

повышение производительности процессоров, как это и было предсказано законом Мура – приводят технологии к порогу развития традиционных вычислительных устройств. От приоритетов бесконечного наращивания тактовой частоты и мощности одного процессора производители переходят к многоядерности, параллелизму и т. п.

На прошедшей в сентябре 2010 года в Абрау-Дюрсо всероссийской научной конференции «Научный сервис в сети Интернет: суперкомпьютерные центры и задачи» (20-25 сентября 2010 г., г. Новороссийск) во многих докладах ставился вопрос: «Что будет при переходе от сегодняшних производительностей суперкомпьютеров в «TeraFlops» к следующему масштабу «ExaFlops»? Вл.В. Воеводин (2010) говорил: «Переход к «ExaScale», естественно, должен будет затронуть такие важнейшие аспекты вычислительных процессов, как: *модели программирования, степень и уровни параллельности, неоднородность программных и аппаратных систем, сложность иерархии памяти и трудности одновременного доступа к ней в распределенных вычислениях, стек системного и прикладного ПО, надежность, энергопотребление, сверхпараллельный ввод/вывод …* ». Все это неизбежно приведёт к смене парадигмы высокопроизводительных вычислений.

Сейчас несколько ядер в процессоре переносного компьютера уже норма, в процессорах суперкомпьютеров ядер уже намного больше. «Джин уже выпущен из бутылки», пройдет совсем немного времени и ядер станет несколько десятков, а потом и тысяч. Появятся совершенно другие архитектуры, ядра будут объединяться в сложные блоки, к данным можно будет получать параллельный одновременный доступ разным вычислительным блокам, “общение” вычислительных блоков между собой будет происходить через общую память. В действительности, изменятся многие аспекты парадигмы: что такое вычислительное устройство и что такое вычислительный процесс. Изменятся традиционные представления о том, как устроен компьютер, что такое вычислительная система. Эти процессы принесут изменения и в стиль программирования, и в то, как будут использоваться вычислительные устройства.

Переход к новой парадигме вычислений приведет, наверное, к тому, что архитектура вычислительных устройств «сдвинется» в сторону «*набора одновременно работающих асинхронных моделей взаимодействующих динамических систем (функциональных элементов)*». Среди новых характерных черт будущей парадигмы все более отчетливо проступают следующие: стохастичность, гибридность, асинхронность, кластерность (отсутствие жесткой централизации и динамическая кластеризация на классы связанных моделей).

Стохастичность. С одной стороны, хорошо известно, что компьютеры становятся все миниатюрнее и миниатюрнее, размер элементарного вычислительного элемента (вентиля) приближается к размеру молекулы или даже атома. На таком уровне законы классической физики перестают работать и начинают действовать квантовые законы, которые в силу принципа неопределенности Гейзенберга принципиально не дают точных ответов о состоянии. С другой стороны, стохастичность – это известное свойство сложных динамических систем, состоящих из огромного числа компонент.

Под *гибридностью* будущих процессов вычислений понимается необходимость рассмотрения комбинации непрерывных и дискретных процессов, т. е. учет непрерывной эволюции протекания физических процессов при работе той или иной модели и скачкообразное переключение с одной модели на другую. Увеличение быстродействия вычислительных устройств и уменьшение их размеров с неизбежностью приводит к необходимости операций с “переходными” процессами, серьёзным ограничением классической модели вычислений является разбиение памяти на изолированные биты, потому как, во-первых, сокращение длины такта и расстояний между битами с определенного уровня делает невозможным рассматривать их изолированно в силу законов квантовой механики. Вместо примитивных операций с классическими битами в будущем было бы

естественно перейти операциям, задаваемыми теми или иными динамическими моделями микромира, оперирующими с наборами взаимосвязанных “битов”. При этом простейшими “моделями” могут остаться классические операции с битами. Обоснованием целесообразности рассмотрения более широкого класса моделей являются успехи в разработках для традиционных сложных многомерных задач новых алгоритмов, работающих “за такт”. Результат получается как итог физического адиабатического процесса. Например, для классической операции с битами – переход физической системы (триггера) из состояния “1” в “0”. П. Шор (1997) предложил алгоритм квантового преобразования Фурье, которое может выполняться за время пропорциональное $(\log N)^2$, а не за $N \log N$ как классическое быстрое преобразование Фурье. В работе Д. Тиена (2003) обсуждается опирающийся на квантовую адиабатическую теорему гипотетически возможный «физический» способ решения за конечное время 10-ой проблемы Гильберта, в работе С.Сысоева и др. (2006) предложен эффективный квантовый алгоритм вычисления “за такт” оценки вектора-градиента многомерной функции, задаваемой с большой степенью неопределенностей. Типичные для математических алгоритмов операции типа “свертки” функций вполне могут обнаружиться “в природе”. Последние исследования похожих моделей показывают, что их выполнение за счет присущей природе способности к самоорганизации не обязательно “раскладывается” на более простые кирпичики, т. е. не всегда может быть записано в виде классического алгоритма. Один из возможных примеров “аналоговой” реализации свертки функций на большом регулярном массиве квантовых точек с характерными размерами до 2 нм представлен в монографии О.Н. Граничина и С.Л. Молодцова (2006).

Асинхронность. Отказ от унифицированных простых вычислительных элементов неизбежно приводит к отказу от

синхронизации работы различных компонент, имеющих существенно отличающиеся физические характеристики и свои длительности “тактов”. В рамках классической теории множеств противоречивый смысл понятия единого “такта” выражается в рамках неразрешимости проблемы континуума в рамках аксиоматики Френкеля-Цермело.

Кластерность. Одним из неожиданных результатов многочисленных попыток в разработках (создании, адекватном описании поведения и управлении) сложных стохастических систем оказалась перспективность модели мультиагентных систем, в которой топология связей агентов между собой меняется со временем. При этом понятию агента может соответствовать как некоторая динамическая модель (компонент системы), так и определенный набор моделей. При отсутствии жесткой централизации такие системы способны эффективно решать достаточно сложные задачи, разбивая их на части и автономно перераспределяя ресурсы на “нижнем” уровне, эффективность часто повышается за счет самоорганизации агентов и динамической кластеризация на классы связанных моделей.

В некотором смысле переход к разработке и созданию моделей сложных вычислений --- закономерный этап развития микропрограммирования. Понимание возможных преимуществ работы устройств с эффективным микрокодом отмечалось еще в 50-е гг. прошлого века. Во втором ряду серии ЕС ЭВМ в конструкцию была заложена возможность динамического микропрограммирования. Хорошо известен целый ряд ЭВМ от “Мир-1”" до “Эльбрус”, использующих языки высокого уровня (HLL) как машинные. В настоящее время *технологии микропрограммирования естественным образом сдвигаются в сторону физических процессов*.

Естественным этапом на этом пути является переход от производства «чистых» процессоров к системам на кристалле (System On Chip, SoC, системы-на-чипе). Сейчас корпорация Intel уже начала производство SoC, в которых на чип перенесены контроллеры ввода-вывода, системы обработки видео и ряд других. В «новых» чипах существенно сокращаются объемы вспомогательных обменов в памяти, повышается эффективность,

производительность и упрощается конструкция плат. Разработчику архитектуры уже меньше надо думать куда «воткнуть» на плате тот или иной узел. Процессоры, а точнее новые SoC, становятся как бы конечными устройствами, которые можно программировать, не задумываясь о периферии. Это идеальная ситуация, когда программисту не надо думать о целой цепочке посредников, через которые должны пройти его команды и сигналы.

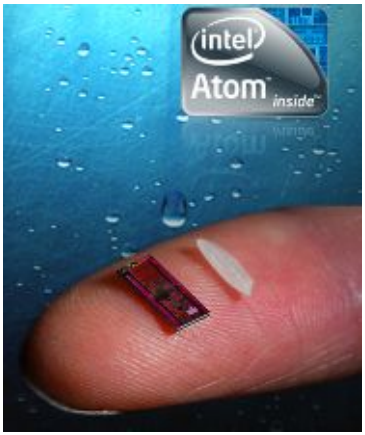
1.2. Компьютерный континуум Intel и процессор Intel-Atom

Основные тенденции полупроводниковой индустрии:

- Закон Мура продолжает действовать.
- Растет стоимость разработки новых технологий и материалов, а также затрат на содержание фабрик.
- Производительность процессоров (систем) также растет; ожидается скачок при переходе на 450 мм пластины.

В 2011 году планируется запуск в Израиле новой фабрики по производству микросхем по технологии 22 нм. Одним из «неожиданных» следствий Закона Мура для развития бизнеса корпорации Интел стал тот факт, что в ближайшее время при очередной смене технологической платформы возможности новых фабрик достигнут такого уровня, что в кратчайшие сроки будет возможно модернизировать все существующие места традиционного использования компьютеров, активно выйти на рынок планшетников, смартфонов и даже встроенных систем. Сохранение при этом универсальной x86 архитектуры и системы команд в значительной степени служит явным конкурентным преимуществом.

Все это приводит к изменению роли и места на рынке корпорации Интел. К традиционным процессорам для серверов и персональных компьютеров и ноутбуков добавляются нетбуки, смартфоны, различные персональные устройства, мобильные Интернет-устройства, «умное» телевидение, бытовая электроника с подключением к Интернет (например, IPTV), встраиваемые системы и т.п. При этом в перспективе нивелируются аппаратные отличия. Так например, уже сейчас для серверов готовятся решения



на основе новых многоядерных процессоров Intel-Atom. Сейчас уже многие говорят о возможности создания своеобразного компьютерного континуума Интел (см. рис. 1.1), включающего наряду с новыми аппаратными платформами и операционные системы, специализированные программы поддержки разработчиков, средства хранения, доступа и реализации разработанных компьютерным сообществом приложений.

Рис. 1.1. Компьютерный континуум Intel®.

Осенью 2010 года корпорация Intel объявила о выпуске семейства «систем-на-чипе» Intel® Atom™ E600 (кодовое наименование Tunnel Creek) для встраиваемых систем и о предстоящем появлении Intel® Atom™ CE4200 (кодовое наименование Groveland) – семейство «систем-на-чипе» III

поколения, базирующееся на архитектуре Intel, предназначеных для использования в «умном» телевидении, в системах, объединяющих стандартное телевещание с Интернетом, библиотекой контента и мощными функциями поиска. В состав решений входят интегрированные ядро Intel Atom™ с частотой 1,2 ГГц и кэш-память второго уровня объемом 512 КБ. Оно предлагает широкие возможности для разработки интерактивных, открытых и персонализированных приложений для запуска на экране телевизора. На базе Intel® Atom™ CE4200 уже разработано решение для потребительской электроники. Эта «Система-на-чипе» осуществляет многопоточное декодирование и обработку HD-видео, поддерживает 3D, MPEG2, MPEG4-2 и VC-11. Решение оснащено интегрированным декодером HD-видео (H.264), позволяет осуществлять видеозвонки, потоковую передачу материалов на другие устройства, в том числе портативную электронику. Благодаря поддержке различных режимов питания новые решения помогают снизить энергопотребление и создавать устройства, удовлетворяющие промышленным стандартам по энергопотреблению. Планы по созданию цифровых приставок нового поколения на базе компонентов Intel озвучили ADB*, Sagemcom,* Samsung* и Technicolor*.

В ближайшее время будет выпушена новая платформа – Medfield (Penwell SoC and Avantele Passage MSIC) – разрабатываемая по технологии 32 нм с существенным уменьшением общих размеров и потребляемой мощности, с размером высокоинтергированной SoC: 144 кв мм, увеличением до 4X graphics performance.

1.3. ОС MeeGo, подсистемы промежуточного слоя (Middleware)

Что общего в разработке приложений для разных мобильных устройств, на что надо особо обратить внимание? Это – ограничения по производительности и энергопотреблению, беспроводное взаимодействие, малые размеры и формы.

С какой основной проблемой сталкиваются сегодня разработчики и пользователи? Все устройства на Рис. 1.2

объединяются тем, что внутри у них стоит процессор Intel® Atom™. У процессоров Intel за последние несколько десятков лет система команд менялась эволюционно, есть преемственность кодов, поколений разработчиков. Но процессоры Intel® Atom™ в каждом из этих устройств используются по-разному, в каждом из устройств он установлен на своей плате, в оригинальном окружении, работает с разными операционными системами. И если вы разрабатываете какое-нибудь свое приложение, например, игру типа Тетрис, то, сделав ее для одного, например, телефона, трудно ее перенести на другое мобильное устройство. Даже при кросс-платформенной среде разработки возможны трудности, связанные с тем, что не все инструкции могут отрабатываться с одинаковым результатом.

Рис. 1.2. Продающиеся сегодня платформы с процессором Intel® Atom™.

Инициативы внутри рынка мобильных телефонов для производителей были чрезвычайно важны в последнее время в связи с бурным ростом этого сегмента рынка. Но на этом рынке долгое время царила фрагментарность. Каждая крупная корпорация стремилась создать свою собственную операционную систему,

разрабатывает свои приложения и живет вместе с несколькими своими союзниками на этом узком рынке, на который очень трудно попасть новым игрокам, разработчикам. Например, если вы студент, аспирант или даже школьник придумали свою хорошую идею, то для ее быстрой реализации очень трудно было «перескочить» через эти установленные барьеры, защищенные патентами, закрытостью кода операционных систем. Если система коммерческая со строгой лицензионной политикой, то, как правило, нельзя деассемблировать ее код, свободно переписывать функции и т. п. Для того, чтобы вставить что-то свое «в середину» чужой системы вам нужно иметь штат хорошо подготовленных юристов, способных подготовить все необходимые соглашения с правообладателями и, конечно, заплатить много денег. Все это, как правило, не реально для молодых и начинающих. Вы с трудом могли пробиться с новой идеей на этот «узкий» рынок.

14 апреля 2010 организацией Linux Foundation было объявлено, что более 25 компаний — Acer, ASUS, BMW Group, Cisco, EA Mobile, Intel. Nokia, Novell, Wind River и др. — пришли к соглашению участвовать в проекте новой операционной системы MeeGo. При этом подчеркивалось, что как платформа с открытым кодом MeeGo поможет снизить фрагментарность рынка и прочие трудности, ускорит внедрение инноваций и вывод на рынок устройств следующих поколений, приложений базирующихся на Интернете, сервисов и достижений пользователей. Бросается в глаза присутствие в списке типичных ИТ-компаний и крупнейшего концерна по производству автомобилей. Идеи интеграции высказывались внутри сообщества производителей мобильных устройств и ранее. О чем периодически докладывалось по итогам ежегодных форумов и выставок. Например, ранее был согласован единый стандарт Mini-USB для подключения блоков питания и компьютеров.

Предполагается, что все эти компании будут по возможности разрабатывать архитектуры своих мобильных устройств под MeeGo. Основная цель – открыть быструю «дорогу» для практической реализации новых идей. Но планируется и достижение серьезного технологического роста за счет того, что единая открытая платформа, близкие архитектуры устройств будут

стимулировать более широкий круг разработчиков. Важной тенденцией является включение в один чип все большего набора функций. Постепенный переход к SoC (системам-на-чипе) естественно будет сокращать разнообразие аппаратных архитектур, что будет соответствовать усилению универсализации.

Новая платформа MeeGo сейчас создается на основе Qt, ofono, Telepathy, Netscape GECKO, Clutter, freedesktop.org, PulseAudio, syncEvolution, LIBSOCIALWEB, conman, X.Org Foundation, gstreamer, cairo, Fennec.

В основе MeeGo лежит ядро Linux с базой данных настроек, системными библиотеками, шиной сообщений и информацией о платформе (см. рис. 1.3).

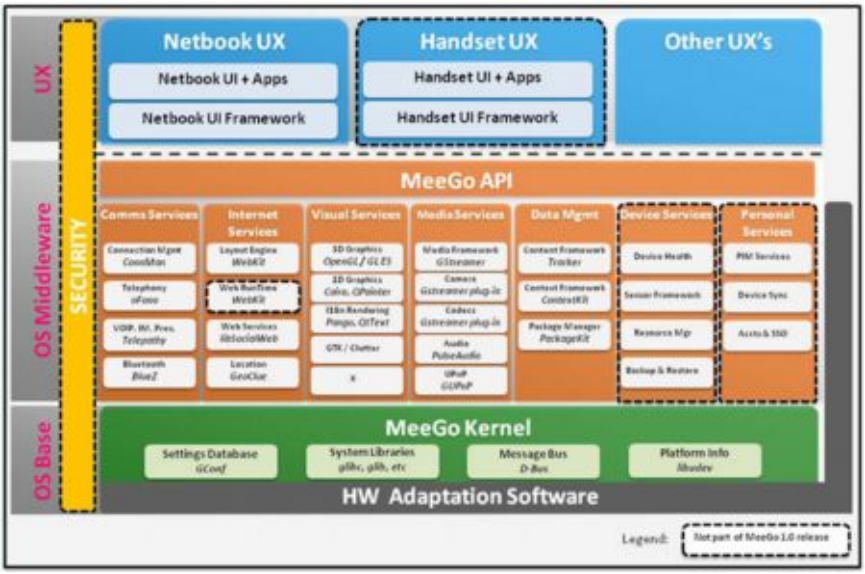


Рис. 1.3. Архитектура ОС MeeGo.

В средний слой входят MeeGo API: сервисы коммуникации (Comms Servises), Интернет-сервисы (Internet Services), сервисы визуализации (Visual Services), Медиа-сервисы (Media Services), управление данными (Data Mgmt), сервисы устройств (Device Services) и персональные сервисы (Personal Services), которые за

исключением двух последних будут детально рассмотрены во второй части курса. Пользовательский слой адресует к конкретным типам устройств пользователей, предоставляя соответствующие приложения и Фреймворки.

ОС MeeGo сейчас разрабатывается для аппаратных платформ со следующими типичными характеристиками:

- Поддержка платформ построенных на процессорах Intel и ARM.
- X86 архитектура процессора с поддержкой четвертого расширения системы команд SSSE3.
- Совместимы графический чипсет Intel.
- Поддержка 3D ускорения.
- Первым двум критериям соответствует ряд процессоров:
- Intel Pentium Dual-Core
- Intel Celeron: 4xx Sequence Conroel-L, Dual-Core E1200, M500 series.
- Intel Xeon: 5300, 5100, 3000.
- Intel 2: Duo, Extreme, Quad.
- Intel i3,i5,i7.
- Intel Atom
- ARM.

Для разработки приложений под MeeGo программист должен быть знаком с подсистемами (библиотеками) промежуточного уровня (слоя программного обеспечения, находящегося между низкоуровневым ПО ядра ОС и пользовательским ПО), обеспечивающими основную функциональность.

Подробный разбор примеров применения средств промежуточного слоя MeeGo можно найти во втором разделе предшествующего курса «Введение в разработку приложений на платформе Atom/MeeGo». Особое внимание в курсе уделено следующим подсистемам:

- Comms Services. Connection Mgmt [ConnMan], Telephony [oFono], VOIP, IM, Pres. [Telepathy], Bluetooth [BlueZ], отвечающим за коммуникацию в мобильных устройствах, начиная от телефонных звонков и отправки SMS сообщений и до обмена мгновенными сообщениями в таких системах как Jabber и IP-телефонии.

- Internet Services. Layout Engine [WebKit], Web Services [libSocialWeb], Location [GeoClue], используемым при работе с Интернетом и для определения местоположения.
- Visual Services. 3D Graphics [OpenGL/GLES], 2D Graphics [Cairo, QPainter], служащих для организации работы с двухмерной и трёхмерной графикой (а именно, OpenGL ES, QPainter и SVG).
- I18n Rendering [Pango, QtText], GTK/Clutter, обеспечивающих интернационализацию (i18n) и локализацию MeeGo приложений.

Data Mgmt. Content Framework [Tracker], Context Framework [ContextKit], Package Manager [PackageKit], предоставляющих возможности по управлению данными пользователя: хранение данных приложения (Publish and Subscribe), управление контактами (библиотека Contacts), работа с визитными карточками (формата vCard) и прочей информации в формате Versit.

1.4. Выводы

Основная цель этой лекции – агитационно-разъяснительная. В ней не было ничего о том, как разрабатывать приложения, но для слушателя и читателя должна была приоткрыться одна из частей горизонта: что и как можно будет делать и использовать в ближайшей перспективе и уже сейчас.

1.5. Контрольные вопросы

1) Какой из перечисленных процессоров потребляет меньше всего энергии?
   1. Intel Celeron Core i5 660.
   2. Intel Celeron Dual-Core E1500.
   3. Intel Atom E620.
   4. Intel Atom Z500.
2) На базе какой из перечисленных систем создавалась ОС MeeGo?

1. MS Windows 7.
2. AIX.
3. Maemo.
4. Solaris.

3) Какие особенности должно иметь оборудование для полноценной работы MeeGo?
   1. Платформа ARM или X86-64, поддержка инструкций SSE3, графический чипсет ATI или Intel с поддержкой 3D ускорения.
   2. Плаформа X86, поддержка SSSE3, графический чипсет GMA-500, ATI, или Nvidia с поддержкой 3D ускорения.
   3. Платформа ARM, поддержка SSE2, графический чипсет GMA без поддержки 3D ускорения.
   4. Платформы ARM или X86, поддержка инструкций SSSE3, поддержка графического чипсета Intel с 3D ускорением.
4) На телефоны какой из перечисленных фирм-производителей сейчас можно установить ОС MeeGo?
   1. Samsung.
   2. Sony-Ericson.
   3. Nokia.
   4. HP.
5) MeeGo – это проект:
   1. Apple.
   2. Microsoft.
   3. Соединение многих проектов.
   4. Новый российский проект.
6) Для каких устройств предназначена ОС MeeGo?
   1. Для серверов.
   2. Для мобильных информационно-развлекательных.
   3. Для хранилищ данных.
   4. Для охранных систем банков.
7) Архитектура ОС MeeGo включает основных слоёв:
   1. Один

2. Три
3. Четыре
4. Шесть

8) Основная новизна ОС MeeGo заключается в:
   1. Красивом графическом интерфейсе.
   2. Свободном доступе к скачиванию.
   3. Лёгком портировании приложений на различные устройства.
   4. Это закрытая информация.

Список литературы

1. Граничин О.Н. Перспективы принципиально новых компьютерных устройств и систем // Суперкомпьютеры, № 2 (6). 2011. С. 8-14.
2. Граничин О.Н., Кияев В.И. Информационные технологии в управлении. --- М.: Изд-во Бином. 2008. 336 с.
3. Воеводин В.В., (2010) Сб. тр. Всерос. научн. конф. «Научный сервис в сети Интернет: суперкомпьютерные центры и задачи» 20-25 сент. 2010 г., Новороссийск – М.
4. Shor, P., (1997). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. 26, 1484-1509.
5. Tien, D.K., (2003) Computing the non-computable, Contemporary Physics, 44, 51-71.
6. Вахитов А.Т., Граничин О.Н., Сысоев С.С., (2006) Точность оценивания рандомизированного алгоритма стохастической оптимизации. Автоматика и телемеханика, № 4 , с. 86-96.
7. Граничин О.Н., Молодцов С.Л. Создание гибридных сверхбыстрых компьютеров и системное программирование. СПб, 2006. 108с.
8. Ibrahim Haddad, Introduction to the MeeGo Project. [On-line] http://wiki.meego.com/images/MeeGo_Introduction.pdf
9. http://www.intel.com/cd/corporate/pressroom/emea/rus/457838.htm
10. http://wiki.meego.com

2. Средства разработки мобильных приложений в Linux

Процесс разработки приложений под Linux на платформе Atom/MeeGo. Среда разработчика, библиотеки, трансляторы, редакторы связей, отладка. Разработка мобильных приложений.

2.1. Введение

Linux является потомком операционных систем семейства Unix, спроектированных максимально просто и лаконично. Unix, а потом и Linux всегда разрабатывались не в одной компании, а в многочисленных лабораториях и институтах по всему миру. В процессе создания и развития Linux постоянно происходил обмен знаниями, идеями, исходным кодом и потому Linux устроен не как монолитная, а как компонентная система. Он изначально спроектирован таким образом, что все компоненты ОС могут разрабатываться разными людьми и быть максимально независимыми друг от друга, что выгодно отличает его от известных коммерческих решений.

ОС Linux создавалась разработчиками для самих себя. Это объясняет удобство разработки программного обеспечения для этой платформы. Среди главных достоинств Linux можно выделить его устойчивость. При сбое и нарушении работы одной из компонент не произойдет отказа системы в целом. Кроме того, не происходит конфликтов и нестабильного поведения в случае, когда сторонние приложения приносят в систему несколько версий одних и тех же компонент. Многие дистрибутивы Linux поставляются со своим менеджером пакетов, что окончательно исключает различного рода проблемы с совместимостью и зависимостью различных модулей.

Архитектура Linux построена прозрачно и логично. Исходный код компонентов операционной системы открыт и хорошо документирован, что позволяет разработчикам принимать активное участие в улучшении качества системы. Кроме того, это облегчает

понимание принципов работы используемого модуля и позволяет намного быстрее подстроиться для работы с ним.

Linux дает разработчику возможность разрабатывать стройный и логичный код, используя разнообразные выверенные временем инструменты для разработки программного обеспечения и переиспользуя уже готовые решения. Специальные пакеты для компиляции и сборки программ в Linux системах позволяют не заботиться о совместимости версий различных компонентов и переносимости программ. Средствами, традиционно используемыми для создания программ для Linux, являются инструменты разработанные в проекте GNU.

Разработка мобильных приложений в Linux является частным случаем кросс-платформенной разработки. При этом используется такие же инструменты, как и для разработки обычных приложений, что позволяет эффективно переиспользовать знакомство с этими инструментами.

2.2. О проекте GNU

Проект GNU был основан Ричардом Столлманом в 1983 году. Его необходимость была вызвана тем, что в то время сотрудничество между программистами было затруднено, так как владельцы коммерческого программного обеспечения чинили многочисленные препятствия такому сотрудничеству. Целью проекта GNU было создание комплекта открытого программного обеспечения под единой лицензией, которая не допускала бы возможности присваивания кем-то эксклюзивных прав на это ПО.

Таким образом, основной задачей проекта стала разработка оболочки, достаточной для использования только открытого программного обеспечения, т.е. разработка новой операционной системы GNU. В настоящее время ядро операционной системы GNU нельзя считать готовым к самостоятельному промышленному использованию. Но операционная система Linux использует многие продукты, разработанные в рамках этого проекта. Прежде всего — это GNU toolchain, который включает в набор необходимых пакетов программ для компиляции и генерации выполняемого кода из исходных текстов программ. GNU toolchain состоит из:

- GNU make: утилита, автоматизирующая преобразование файлов из одной формы в другую. Чаще всего это компиляция исходного кода в объектные файлы и дальнейшая компоновка в исполняемые файлы и библиотеки.
- GNU Compiler Collection (GCC): набор компиляторов проекта GNU.
- GNU Binutils: набор инструментов для управления объектными файлами.
- GNU Bison: программа, предназначенная для автоматического создания синтаксических анализаторов по заданному описанию грамматики.
- GNU m4: язык макроопределений.
- GNU Debugger (GDB): отладчик проекта GNU.
- GNU build system (autotools): утилиты для сборки и компиляции исходного кода. Состоит из autoconf, automake, autoheader и libtool.

Рассмотрим наиболее значимые и частоиспользуемые программистами инструменты, разработанные в проекте GNU и являющиеся неотъемлемой частью дистрибутивов операционной системы Linux.

2.2.1. Описание GNU autotools

GNU autotools состоит из некоторого стандартного набора утилит, которые предназначены для компиляции исходного кода программы под целевую платформу. При помощи них автоматизированно решается задача переносимости кода в различных Unix-системах. Переносимость кода — одна из сложных проблем, которую надо решать при разработке программного обеспечения. Компиляторы языка C, например, могут существенно отличаться друг от друга. Некоторые стандартные функции могут быть пропущены, иметь иное имя или объявляться в другом заголовочном файле. Все эти ситуации могут быть обработаны путем заключения различных кусков кода в директивы для препроцессора типа #if, #ifdef и других. Но это обязывает разработчика предусматривать огромное количество вариантов

систем и тщательно продумывать, как код будет на них выполняться.

Набор autotools был разработан для устранения этой проблемы. Он вызывается конечным пользователем в виде комбинации команд: ./configure && make && make install. Autotools состоит из трех основных компонент: autoconf, automake и libtool.

Утилита autoconf

Autoconf — это утилита для создания скриптов командного процессора, которые автоматически конфигурируют пакеты с исходным кодом так, чтобы они могли работать на множестве UNIX-подобных систем.

Утилита autoconf создает сценарий установки для включения их в распространяемый код, который по умолчанию называется configure. На целевой машине он выполняется независимо и не требует инсталляции утилиты в системе. Метод установки программного обеспечения при помощи сценария configure получил широкое распространение и хорошо знаком многим пользователям программ с исходным кодом. Для установки ПО, пакетизированного при помощи утилиты autoconf, как правило, необходимо выполнить команды:

```
./configure
make
make install
```

Использование утилиты autoconf позволяет переносить приложение практически на любую Unix-систему. Сценарий configure проверяет некоторые системные возможности целевой платформы и формирует компоновочные make-файлы, учитывающие возможности текущей среды.

В зависимости от сложности приложения и требуемой степени его переносимости процесс создания установочных сценариев может изменяться от достаточной простой процедуры до сложной. В качестве общего руководства можно использовать последовательность действий, приведенную в книге А. Гриффитса.

Утилита automake

Automake — это утилита для автоматического создания файлов ‘Makefile.in’ из файлов ‘Makefile.am’. Каждый файл ‘Makefile.am’ фактически является набором макросов для программы make. Типичный входной файл Automake является просто набором макроопределений.

2.2.2. Описание GNU make

Утилита GNU make разработана для сборки исходного кода и компиляции его в объектные файлы. Для того чтобы понять необходимость в подобном инструменте рассмотрим несложную программу на С. Пусть программа prog состоит из пары файлов кода main.c и supp.c и используемого в каждом из них файла заголовков defs.h. Тогда для создания prog необходимо из пар (main.c defs.h) и (supp.c defs.h) создать объектные файлы main.o и supp.o, а затем слинковать их в prog. При сборке вручную, выйдет что-то вроде:

```
cc -c main.c defs.h
cc -c supp.c defs.h
cc -o prog main.o supp.o
```

Если впоследствии заголовочный файл defs.h будет изменен, то нам потребуется полная перекомпиляция; а если изменится файл supp.c, например, то перекомпиляцию файла main.o делать не нужно. Отсюда возникает желание для каждого файла, который должен получиться в процессе компиляции, указать, на основе каких файлов и с помощью какой команды он создается. Таким образом, нам необходима программа, которая собирает правильную последовательность команд, необходимую для получения требуемых результирующих файлов, и которая запускает процесс создания требуемого файла только если такого файла не существует или он старше чем файлы, от которых у него есть зависимости. Именно эта функциональность и реализована в утилите make.

Всю информацию о проекте make получает из Makefile, который обычно хранится в том же каталоге, что и исходные тексты программы. Простейший Makefile состоит из синтаксических конструкций двух типов: целей и макроопределений.

Цели в Makefile - это файлы, которые предполагается получить после компиляции проекта. Описание цели состоит из трех частей: имени цели, списка зависимостей и списка команд интерпретатора sh, требуемых для построения цели. Имя цели - непустой список файлов, которые предполагается создать. Список зависимостей - список файлов, из которых строится цель. Примером простого Makefile может послужить уже упоминавшаяся программа prog:

```
prog: main.o supp.o
cc -o prog main.o supp.0
main.o supp.o defs.h
```

В рассмотренном Makefile одни и те же объектные файлы перечисляются несколько раз. Для упрощения таких ситуаций make поддерживает макроопределения.

Макроопределение представляет из себя пару «переменная-значение ». Значение может являться произвольной последовательностью символов, включая пробелы и обращения к значениям уже определенных переменных. Теперь при обращении к переменной-макроопределению в Makefile вместо нее будет подставлено ее текущее значение. Обращение к значению переменной при этом выглядит как $(переменная). Учитывая это, перепишем наш Makefile:

```
OBJS = main.o supp.o
prog: $(OBJS)
cc -o prog $(OBJS)
$(OBJS): defs.h
```

Makefile пишется таким образом, чтобы запуск команды make приводил к компиляции проекта. Но кроме компиляции Makefile может использоваться и для других целей, например, для очистки проекта от результатов компиляции или для вызова процедуры

инсталляции проекта в системе. Для выполнения подобных действий в Makefile могут быть указаны дополнительные цели, обращение к которым будет осуществляться указанием их имени аргументом вызова make (например, «make install»). Подобные вспомогательные цели носят название фальшивых, что связанно с отсутствием в проекте файлов, соответствующих их именам. Фальшивая цель может содержать список зависимостей и должна содержать список команд для исполнения. Поскольку она не имеет соответствующего файла в проекте, при каждом обращении к ней make будет пытаться ее построить. Однако, возможно возникновение конфликтной ситуации, когда в каталоге проекта окажется файл с именем, соответствующим имени фальшивой цели. Если для данного имени не определены файловые зависимости, он будет всегда считаться актуальным (up to date) и цель выполняться не будет. Для предотвращения таких ситуаций утилита make поддерживает встроенную переменную .PHONY, которой можно присвоить список имен целей, которые всегда должны считаться фальшивыми.

Примерами фальшивых целей можно назвать: all, clean и install. Цель all обычно используется как псевдоним для сборки сложного проекта, содержащего несколько результирующих файлов (исполняемых, разделяемых библиотек, страниц документации и т.п.). Цель clean используется для полной очистки каталога проекта от результатов компиляции и мусора, например, резервных файлов, создаваемых текстовыми редакторами. Цель install используется для инсталляции проекта в операционной системе.

2.2.3. Описание GNU Compiler Collection

GCC — это компилятор проекта GNU, первый вариант которого был реализован Ричардом Столлманом в 1985 году. На данный момент GCC поддерживает следующие языки программирования: Ada, C, Objective-C, C++, Fortran, Java. Также он является абсолютным лидером по количеству поддерживаемых процессоров и операционных систем.

Внешний интерфейс GCC является стандартом для компиляторов в Unix-системах. Пользователь вызывает управляющую программу gcc. Команда gcc интерпретирует аргументы командной строки, определяет и выполняет для каждого входного файла свои компиляторы нужного языка, запускает, если необходимо, ассемблер и компоновщик.

Работа компилятора gcc состоит из трех этапов: обработка препроцессором, компиляция и компоновка (или линковка).

Препроцессор включает в основной файл содержимое всех заголовочных файлов, указанных в директивах #include. В заголовочных файлах обычно находятся объявления функций, используемых в программе, но не определённых в тексте программы. Их определения находятся где-то в другом месте: или в других файлах с исходным кодом или в бинарных библиотеках.

Вторая стадия – компиляция. Она заключается в превращении текста программы на исходном языке в набор машинных команд. Результат сохраняется в объектном файле.

Последняя стадия – компоновка. Она заключается в связывании всех объектных файлов проекта в один, связывании вызовов функций с их определениями, и присоединением библиотечных файлов, содержащих функции, которые вызываются, но не определены в проекте. В результате формируется запускаемый файл.

Все опции командной строки можно разделить на три категории:

1. Специфичные к языку. Компилятор GCC разработан для компиляции нескольких языков и некоторые опции применяются только к одному или двум из них.
2. Специфичные к платформе. Как уже упоминалось выше, компилятор GCC является абсолютным лидером по количеству поддерживаемых платформ. Поэтому некоторые опции применяются только когда создается объектный код для конкретной целевой платформы. К примеру, если целевой платформой выбрана Intel 386, для того чтобы определить, что числа с плавающей точкой, возвращаемые вызываемыми функциями, должны сохраняться в аппаратных регистрах с плавающей точкой может быть использован

набор опций -fp -ret -in -387.

3. Общие. Многие опции имеют общее значение для всех языков программирования и аппаратных платформ. Например, опция -o указывает компилятору оптимизировать выводимый объектный код.

Команда-драйвер gcc обрабатывает все известные ей опции, а оставшиеся передает процессу, компилирующему конкретный язык. Если опция компилируемого языка не известна ему, то будет выдано сообщение об ошибке.

Компилятор gcc развивается весьма динамично. Каждые несколько минут в экспериментальную версию проекта вносятся разнообразные изменения: исправляются найденные ошибки, добавляется новая функциональность. Домашняя страничка компилятора находится по адресу www.gnu.org/software/gcc/gcc.html. На ней можно отслеживать текущую версию продукта и следить за сделанными изменениями.

2.2.4. Описание GDB

Стандартным средством для отладки программ, скомпилированных компилятором GCC, является отладчик GDB. Этот отладчик свободно распространяется в рамках проекта GNU. Домашняя страничка отладчика находится по адресу www.gnu.org/software/gdb/gdb.html.

Чтобы указать компилятору gcc, что вы планируете отлаживать вашу программу, и поэтому нуждаетесь в дополнительной информации, добавьте ключ -g в опции компиляции и компоновки. Например, если программа состоит из двух файлов main.c и utils.c, можно скомпилировать ее командами:

```
gcc -c -g Wall main.c
gcc -c -g -Wall utils.c
gcc -g -o prog main.o utils.o
```

или одной командой:

```
gcc -g -Wall -o prog main.o utils.o
```

Обе последовательности команд приводят к созданию исполняемого файла prog.

Чтобы выполнить полученную программу под управлением gdb, введите

```
gdb prog
```

Вы увидите командное приглашение GDB:

```
(gdb)
```

Это очень простой, но эффективный текстовый интерфейс отладчика. Его вполне достаточно, чтобы ознакомиться с основными командами gdb.

Когда GDB запускается, ваша программа в нем еще не выполняется; вы должны сами сообщить GDB, когда ее запустить. Как только программа приостанавливается в процессе выполнения, GDB ищет определенную строку исходной программы с вызовом определенной функции - либо строку в программе, где произошел останов, либо строку, содержащую вызов функции, в которой произошел останов, либо строку с вызовом функции и т.д. Далее используется термин текущее окно, чтобы сослаться на точку останова.

Как только возникает командное приглашение, можно использовать следующие команды:

help command — выводит краткое описание команды GDB (просто help выдает список доступных разделов справки);

run command-line-arguments — запускает программу с определенными аргументами командной строки. GDB запоминает переданные аргументы, и простой перезапуск программы с помощью run приводит к использованию этих аргументов;

where — создает цепочку вызовов функций, произошедших до попадания программы в текущее место. Синонимом является команда bt;

up — перемещает текущее окно так, чтобы GDB анализировал место, из которого произошел вызов данного окна. Очень часто Ваша программа может войти в библиотечную функцию — такую, для которой не доступен исходный код, например, в процедуру ввода-вывода. вам может понадобиться несколько команд up, чтобы перейти в точку программы, которая была выполнена последней;

down — производит эффект, обратный up;

print E — выводит значение E в текущем окне программы, где E является выражением C++ (обычно просто переменной). Каждый раз при использовании этой команды, GDB нумерует ее упоминание для будущих ссылок.

quit — выход из GDB;

Ctrl-c — если программа запущена через оболочку shell, Ctrl-c немедленно прекращает ее выполнение. В GDB программа приостанавливается, пока ее выполнение не возобновится;

break place — установить точку останова; программа приостановится при ее достижении. Простейший способ - установить точку останова после входа в функцию.

Команда break main остановит выполнение в начале программы. Также можно установить точки останова на определенную строку исходного кода. Когда программа запущена и она достигнет точки останова, то об этом выводится специальное сообщение.

delete N — удаляет точку останова с номером N. Если опустить N, будут удалены все точки останова;

cont или continue — продолжает обычное выполнение программы;

step — выполняет текущую строку программы и останавливается на следующем операторе для выполнения;

next — похожа на step, однако, если текущая строка программы содержит вызов функции (так что step должен будет остановиться в начале функции), не входит в эту функцию, а выполняет ее и переходит на следующий оператор;

finish — выполняет команды next без остановки, пока не достигнет конца текущей функции.

Помимо текстового интерфейса для отладчика dbg существует различные графические оболочки, например, DataDisplayDebugger (DDD). Эта оболочка является надстройкой над текстовыми отладчиками, реализующей для них удобный графический

интерфейс. Программа DDD также входит в проект GNU. Ее домашняя страничка находится по адресу www.gnu.org/software/ddd. DataDisplayDebugger работает в среде X-Windows.

GDB предоставляет обширные возможности для слежения и контроля выполнения компьютерных программ. Он может выполнять действия четырех основных типов (а также другие, дополняющие эти основные), чтобы помочь вам выявить ошибку:

- Начать выполнение вашей программы, задав все, что может повлиять на ее поведение.
- Остановить вашу программу при указанных условиях.
- Исследовать, что случилось, когда ваша программа остановилась.
- Изменить вашу программу, чтобы вы могли поэкспериментировать с устранением эффектов одной ошибки и продолжить выявление других.

Сейчас GDB активно развивается. В версии 7.0, например, добавлена поддержка «обратимой отладки», позволяющей отмотать назад процесс выполнения, чтобы посмотреть, что произошло. Также в версии 7.0 была добавлена поддержка скриптинга на Python.

2.3. Инструменты Intel для разработки под Linux

Корпорация Intel предлагает свой большой набор инструментов для разработки под Linux, наиболее важными среди которых являются:

- Intel C++ Compiler — оптимизирующий компилятор языков C\C++ для платформы x86. icc использует особые возможности и преимущества процессоров Intel и в большинстве случаев даёт значительно более производительный код в сравнении с gcc. Значительным преимуществом icc является то, что он использует синтаксис командной строки, похожий на синтаксис командной строки gcc. Это позволяет переводить

достаточно большие проекты, использующие gcc, на компиляцию icc, без особых трудозатрат. В частности, компилятором icc было успешно скомпилировано ядро Linux.

- Intel® VTune™ Performance Analyzer — профилировщик. Позволяет изучить производительность отдельных участков кода и общую производительность, выявляя узкие места. В отличие от инструментов, симулирующих выполнение кода на виртуальном процессоре, VTune™ выполняет код на CPU от Intel, используя для измерений многочисленные отладочные регистры процессора. В этом VTune™ аналогичен OProfile, однако работа с последним значительно менее наглядна и не вполне раскрывает богатые отладочные функции процессоров Intel.

2.4. Свободные IDE для разработки программного обеспечения на C/C++ под Linux

В UNIX/Linux есть большое количество инструментов для разработки проектов на C/C++. Некоторые разработчики предпочитают традиционные Vi/Emacs/утилиты командной строки, а другие — современные средства разработки.

При создании больших проектов, использование средств разработки особенно оправдывает себя. Как правило, они дают разработчику возможности автоматического дополнения кода, свертывания кода, подсветки синтаксиса, предоставляют шаблоны кода, встроенный компилятор и отладчик. В особенности он помогает людям справиться с несколькими файлами при использовании GUI, в отличие от утилит командной строки или традиционных редакторов без GUI.

Перечислим наиболее популярные IDE для разработки на C/C++ под Linux:

- QtCreator — кроссплатформенная IDE для работы с фреймворком *Qt*, разработанная *Qt* Software.IDE для работы с фреймворком *Qt*, разработанная *Qt* Software. Эта IDE была специально разработана для работы с *Qt*, имеет возможности расширения плагинами, встроенный QtDesigner и QtAssistant и графический фронтенд для gdb.
- NetBeans — свобаодная интегрированная среда разработки приложений на языках программирования Java, JavaFX, Ruby, Python, PHP, JavaScript, C++, Ada и другие. NetBeans поддерживает рефакторинг, профилирование, выделение синтаксических конструкций цветом, автодополнение набираемых конструкций на лету, множество переопределенных шаблонов кода, удаленную отладку и др. В NetBeans поддерживается UML, SOA, языки программирования Ruby, а также средства для создания приложений на J2ME для мобильных телефонов. NetBeans IDE поддерживает плагины, позволяя разработчикам расширять возможности среды.
- Eclipse — в первую очередь платформо-независимая Java IDE, нацеленная на групповую разработку: среда интегрирована с системами управления версиями — CVS в основной поставке, для других систем (например, Subversion, MS SourceSafe) существуют плагины. Второе назначение Eclipse — служить платформой для разработки новых расширений, чем он и завоевал популярность: любой разработчик может расширить Eclipse своими модулями. Уже существуют C/C++ Development Tools (CDT), разрабатываемые инженерами QNX совместно с IBM, и средства для языков COBOL, FORTRAN, PHP и пр. от различных разработчиков. Множество расширений дополняет среду Eclipse менеджерами для работы с базами данных, серверами приложений и др.
- Anjuta — это гибкая интегрированная среда разработки (Integrated Development Environment — IDE) для языков C и C++ в GNU/Linux, которая была написана для GTK/GNOME и включает ряд мощных средств для программирования. Среди

них — средства управления проектом, мастера приложений, встроенный интерактивный отладчик, мощный редактор исходного кода со средствами просмотра и подсветкой синтаксиса.

- Kdevelop — свободная среда разработки программного обеспечения для Unix-подобных систем. Она поддерживает подсветку исходного кода с учетом синтаксиса используемого языка программирования; менеджер проектов, для проектов разного типа, таких как Automake, qmake для проектов базирующихся на Qt и Ant для проектов, базирующихся на Java; навигатор классов (Class Browser); front-end для GNU Compiler Collection; front-end для GNU Debugger; wizards для генерации и обновления определения классов и framework; автоматическую систему завершения кода (Си/C++); встроенную поддержку Doxygen; систему контроля версий.

2.5. Инструменты профилировки и отладки

Для разработки эффективного программного обеспечения часто приходится выполнять профилирование кода, которое включает в себя сбор характеристик работы программы,таких как время выполнения отдельных фрагментов (обычно подпрограмм), число верно предсказанных условных переходов, число кэш промахав и многое другое.

Перечислим основные средства для профилирования программ при разработке под ОС Linux:

- GNU profiler (gprof) используется для того, чтобы определить, сколько времени уходит на выполнение той или иной части программы, как часто вызываются те или иные процедуры; для использования gprof необходимо компилировать программу со специальными опциями для включения «профилирования».
- Valgrind – инструментальное программное обеспечение, предназначенное для отладки утечек памяти, профилировки, построения дерева вызовов.
- KCacheGrind – графический анализатор вывода Valgrind .

- strace – утилита, выполняющая трассировку системных вызовов
- OProfile – профилировщик, использующий счётчики производительности процессора.

2.6. Разработка мобильных приложений

Разработка мобильных приложений – частный случай так называемой кросс-разработки. В случае кросс-разработки платформа, для которой пишется приложение (целевая платформа, например, Linux ARM в виде смартфона) отличается от платформы, на которой работает программист и создается исполняемый код для целевой платформы (платформа разработчика, например, Linux x86 в виде обычного PC). В случае ОС MeeGo аппаратная часть платформы может быть одинакова – x86 – но способ разработки все равно останется кроссплатформенный. Мы рассмотрим четыре составляющих кросс-платформенной разработки – создание исполняемого кода, его запуск, отладка и интегрированные среды разработчика.

Наиболее важной частью кросс-платформенной разработки является набор инструментов для создания исполняемого кода. В него входит компилятор, компоновщик, другие утилиты, набор библиотек для целевой платформы – этого достаточно, чтобы создать исполняемый файл для целевой платформы. Обычно этот набор инструментов является частью или близким родственником GNU Compiler Collection, то есть применяются те же названия, те же ключи командной строки, те же приемы использования. Ниже приведен пример набора инструментов для архитектуры ARM:

```
[r...@v...]# ls -1 toolchain/arm-eabi-4.4.0/bin/
arm-eabi-ar
arm-eabi-as
arm-eabi-g++
arm-eabi-gcc
arm-eabi-gdb
arm-eabi-gprof
arm-eabi-ld
arm-eabi-objdump
```

```
arm-eabi-ranlib
arm-eabi-strings
arm-eabi-strip
```

Для управления процессом сборки, документирования и т.п. можно использовать те же средства, что и для обычной разработки под Linux.

Запуск объектного кода может может быть осуществлен разными способами. Самый естественный – запуск непосредственно на целевом устройстве. Для этого требуется само целевое устройство, подключенное к компьютеру разработчика по USB или Ethernet. Это наиболее надежный способ, но не всегда самый удобный. Финальное тестирование необходимо производить именно на целевом устройстве.

Альтернатива – установить целевую операционную систему на универсальную виртуальную машину на платформе разработчика. В качестве такой машины для MeeGo рекомендуется Virtual Box, также можно использовать VMWare. При этом надо учитывать, что по некоторым параметрам виртуальная машина может отличаться от реального устройства – например, по скорости работы, особенностям периферии и т. п.

Промежуточным вариантом является использование предоставляемого производителем целевой платформы эмуляторы. Фактически это разновидность виртуальной машины, которая, с одной стороны, максимально адаптирована для эмуляции целевой платформы, а с другой стороны, интегрирована со средствами разработки, например, IDE, которые также предоставляются производителем.

В настоящее время для запуска обычно не ограничиваются переносом на целевую платформу одного исполняемого файла, а формируют целый пакет со всеми зависимостями и правилами установки (например, APK для Android или RPM для MeeGo), который затем устанавливается на целевой платформе. С помощью такого же пакета финальная версия приложения попадает на устройства конечных пользователей. Это, с одной стороны, увеличивает переносимость приложения и облегчает запуск сложных приложений, с другой – слегка усложняет и удлиняет процедуру запуска.

При кросс-платформенной разработке процедура отладки приложения изменяется не сильно. Описанный выше gdb можно использовать и для кросс-отладки. Для этого на целевой платформе отлаживаемое приложение запускается из-под gdbserver:

```
gdbserver host:port exe [args …]
```

На платформе разработчика запускается gdb и в его консоли устанавливается связь с gdbserver:

```
target remote host:port
```

gdb устанавливает соединение с gdbserver согласно указанным host:port и после этого программист может проводить сеанс отладки как в обычном gdb. При использовании интегрированной среды разработки все эти детали взаимодействия gdb скрыты от программиста – зачастую, глядя на экран платформы разработчика, невозможно определить, происходит отладка локально или на целевой платформе.

Еще одним популярным способом отладки при кросс-платформенной разработке является полный отказ от gdb и интенсивное использование логов.

Современные IDE, такие как QtCreator или Eclipse, поддерживают кросс-платформенную разработку. Эта поддержка может быть встроенной или реализовываться в виде плагинов. В любом случае IDE берут на себя львиную долю огранизационной работы, освобождая программисту время для творчества. Внешне процесс кросс-разработки с использованием IDE практически не отличается от обычного.

2.7. Выводы

Целью этой лекции было дать представление о разработке под ОС Linux, упомянуть наиболее известные инструменты для разработки и коротко охарактеризовать их. Разумеется, дать полное описание программ и руководство по их использованию в рамках небольшого доклада невозможно. Вся дополнительная информация по упомянутым средствам для разработки, отладки,

профилирования кода может быть найдена в Интернете, на страницах Википедии и официальных сайтах продуктов.

Преимущества разработки под ОС Linux заключается еще и в том, что любая информация об интересующей вас программе может быть найдена легко. Документация по основным инструментам ОС Linux, в частности по основным инструментам разработки и отладки, хорошо структурирована и отвечает общепринятым в среде открытого программного обеспечения стандартам.

2.8. Контрольные вопросы

1) К преимуществам разработки под ОС Linux можно отнести:
   1. производительность выполнения кода на целевой платформе Linux намного выше, чем на других известных
   2. только в Linux поддерживается динамическая линковка библиотек
   3. для Linux существует большое количество библиотек с открытым кодом
2) Какую цель ставил изначально перед собой проект GNU?
   1. создать свободную операционную систему
   2. создать большое количество инструментов для разработки в ОС Linux
   3. разработать ядро Unix-подобной ОС
   4. создать компилятор gcc
3) Как расшифровывается аббревиатура GNU?
   1. Got No Use
   2. это не аббревиатура, а название животного
   3. GNU's Not UNIX
   4. Great New Utility
4) Кто является основателем проекта GNU?
   1. Марк Шатлворт
   2. Стив Балмер
   3. Линус Торвальдс
   4. Ричард Столлман
5) Когда был начат проект GNU?

1. 1989
2. 1983
3. 1995
4. 2001

6) Укажите, что из списка было разработано не в рамках проекта GNU:
   1. Qt
   2. gcc
   3. glibc
   4. GNOME

7) gcc – это:
   1. отладчик
   2. компилятор для языка Python
   3. компилятор, поддерживающий такие языки как C, C++, Java, Ada, Objective C, Objective C++, Fortran
   4. профилировщик

8) В каком году состоялся первый релиз gcc?
   1. 1987
   2. 1983
   3. 1991
   4. 2002

9) g++ - это:
   1. обёртка GCC для компиляции и линковки исходных файлов на C++
   2. Ответвление от проекта GCC, имеющее целью создать компилятор для языка OCaml
   3. обёртка GCC для компиляции и линковки исходных файлов на Fortran

10) Пусть исходный код Вашего приложения состоит из нескольких модулей: file1.c file2.c file3.c

Для того чтобы выполнить компиляцию приложения необходимо выполнить следующую команду:

   1. gcc -o myprogram file1.c gcc -o myprogram file2.c gcc -o myprogram file3.c
   2. gcc -o myprogram file1.c file2.c file3.c
   3. gcc -o file1.c file2.c file3.c

11) Куда по умолчанию скопмпилируется исполняемый код при

вызове программы gcc mysource.c?

1. будет сгенерирована ошибка
2. a.out
3. вывод генерируется в стандартный поток вывода
4. Makefile

12) Линковка с внешними библиотеками выполняется при помощи опции:
    1. -l
    2. -o
    3. -c
    4. -static
13) Какой из следующих ключей указывает gcc не выполнять линковку?
    1. -l
    2. -o
    3. -c
    4. -L
14) Название файла, который будет создан в результате компиляции и, возможно, линковки, задаётся после ключа
    1. -l
    2. -o
    3. -c
    4. -Wall
15) Что не входит в систему автоматической сборки GNU?
    1. gdb
    2. automake
    3. make
16) Что выполняет утилита make?
    1. генерирует shell-скрипт, собирающий данные о системе, на которой выполняется сборка
    2. генерирует Makefile
    3. выовдит информацию о системе
    4. осуществляет сборку
17) Что выполняет утилита autoconf?
    1. выполняет конфигурацию пользовательской системы, устанавливая недостающие пакеты
    2. генерирует скрипт configure из файла configure.ac

3. осуществляет сборку
4. генерирует Makefile

18) Что выполняет утилита automake?
1. осуществляет сборку
2. генерирует Makefile.in на основании Makefile.am и configure.ac
3. собирает данные о системе, в которой производится сборка

19) Что содержит Makefile?
1. shell-скрипт, собирающий данные о системе
2. непосредственные указания для сборки проекта
3. указания для пользователя, как собрать и установить пакет
4. двоичный код — результат сборки

20) В стандартной модели использования autotools Makefile генерируется автоматически:
1. скриптом configure из файла Makefile.am
2. скриптом configure из файлов Makefile.in config.h.in
3. программой make из файла Makefile.am
4. стандартная модель использования autotools предполагает написание Makefile вручную

21) Что представляет из себя Makefile.am?
1. такого файла нет
2. данные о структуре и внутренних зависимостях исходного кода
3. shell-скрипт, собирающий данные о системе, на которой выполняется сборка
4. служебные данные, генерируемые в процессе работы скрипта configure

22) Укажите, какая из задач не решается системой автоматизированной сборки:
1. упрощение сборки больших проектов
2. отслеживание ошибок в коде
3. отслеживание зависимостей между модулями
4. выполняется компиляция только обновлённых частей проекта

23) POSIX – это:
1. набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой

2. система контроля версий
3. скрипт для сбора данных о системе, на которой выполняется сборка
4. UNIX-подобная ОС

24) Какое из следующих определений наиболее точно соответствует цели в правилах Makefile?
1. имя исполняемого файла, который мы получим в конце сборки
2. имя файла, который является результатом одного из этапов сборки
3. узел в дереве зависимостей, по которому осуществляется сборка
4. наименование целевой архитектуры, на которой производится сборка

25) Какое из следующих определений наиболее точно соответствует реквизитам в правилах Makefile?
1. список целей, от которых зависит другая цель
2. макроопределение, задающее название цели в виде внутренней переменной
3. shell-скрипт, собирающий цель
4. список файлов, из которых компилируется некий другой файл

26) В каком случае требуется пересборка цели?
1. если один из реквизитов был обновлён ранее цели
2. если цель является .PHONY-целью

27) Что произойдет, если файл, заданный в качестве цели для некоторого правила, существует в проекте и в правиле не определены реквизиты?
1. файл будет удален
2. файл будет считаться актуальным и цель пересобираться не будет
3. цель всегда будет пересобираться
4. работа make завершится с ошибкой

28) С чем связано название некоторых целей в Makefile фальшивыми:
1. с отсутствием в проекте файлов, соответствующих их именам

2. с тем, что они заданы неявно

29) Для чего нужны фальшивые цели в Makefile?
1. задания макроопределений
2. инсталляции проекта в системе
3. удаления результатов сборки
4. выполнения вспомогательных действий, напрямую не связанных с созданием каких-либо файлов

30) Неявно заданное правило — это:
1. правило, имеющее phony-цель
2. правило, создаваемое автоматически в соответствии с мета-правилом, где задана не конкретная цель, а некий паттерн имени цели
3. правило, использующее в командах переменные.
4. правило, в котором задана цель и реквизиты, но не задано никаких команд.

31) Для чего нужна встроенная переменная .PHONY?
1. в этой переменной указывается список имен целей, которые всегда должны считаться фальшивыми
2. в этой переменной указывается список имен целей, которые никогда не должны пересобираться
3. в этой переменной хранятся общие реквизиты для нескольких целей

32) Что из нижеперечисленного **не** входит в функции системы управления пакетами?
1. управление процессом установки компонентов ПО
2. управление процессом удаления компонентов ПО
3. управление процессом установки компонентов ПО из исходного кода
4. управление полномочиями пользователей в системе

33) Какой командой, используя систему управления пакетами apt, можно скачать пакет исходного кода?
1. apt-get source <название пакета>
2. apt-get build-dep <название пакета>
3. aptitude <название пакета>
4. yum install <название пакета>

34) Какой командой, используя систему управления пакетами apt, можно установить пакеты, необходимые для сборки данного

пакета из исходного кода?

1. apt-get source <название пакета>
2. apt-get build-dep <название пакета>
3. make depcomp
4. Достаточно установить пакет при помощи apt-get install. Необходимые пакеты будут установлены автоматически

35) Что из перечисленного не является системой автоматизации сборки?
    1. qmake
    2. Qt
    3. Cmake
    4. Scons
36) Как расшифровывается аббревиатура gdb?
    1. Greedy Disk-Conserving Broadcasting
    2. Genome Database
    3. GNU database
    4. GNU debugger
37) Кем и когда была выпущена первая версия gdb?
    1. Ричардом Столлманом в 1986
    2. Джоном Джилмором в 1992
    3. Марком Шатлвортом в 2001
38) Отладку какого из перечисленных языков не поддерживает отладчик gdb?
    1. C
    2. Fortran
    3. Modula-2
    4. Haskell
39) Какой из перечисленных ниже инструментов предназначен для обнаружения утечек памяти в коде?
    1. KcacheGrind
    2. perfmon
    3. Valgrind
    4. strace
40) Какой инструмент требует перекомпиляции ядра для своей работы?
    1. KcacheGrind
    2. perfmon

3. Oprofile
4. PTLsim

41) Какой инструмент позволяет симулировать работу процессора семейства x86?
1. PTLsim
2. KcacheGrind
3. gprof
4. strace

Список литературы

1. Гриффитс А. GCC. Настольная книга пользователей, программистов и системных администраторов. – М., 2004. – 624 с.
2. Wikipedia
3. Документация по GCC (http://gcc.gnu.org/onlinedocs/)
4. Документация по GNU make (http://www.gnu.org/software/make/manual/make.html)
5. Autobook AKA “The Goat Book” (http://sources.redhat.com/autobook/)
6. Документация по automake (http://sources.redhat.com/automake/automake.html)
7. Документация по autoconf (http://www.gnu.org/software/autoconf/manual/autoconf.html)
8. Документация по GDB (http://www.gnu.org/software/gdb/documentation/)
9. Rehman R.U., Paul C. The Linux Development Platform.

3. MeeGo SDK. Обзор технологии Qt. Магазин приложений AppUp

Nokia *Qt* SDK – инструмент для кросс-платформенной разработки. Интеловские инструменты для разработки программного обеспечения для мобильных устройств и элементы технологии разработки. Магазин приложений Intel AppUp.

3.1. Введение

Основная цель лекции – показать способы разработки под операционную систему MeeGo и дать небольшой обзор технологии *Qt*, которая представляет из себя кросс-платформенный инструментарий для разработки приложений, и которая является основой API ОС MeeGo.

Разработка под ОС MeeGo начинается с развертывания MeeGo SDK (software development kit), которое может осуществляться несколькими способами. В первой части нашей лекции мы обсудим какие именно варианты развертывания SDK предлагаются на сегодняшний день, выявим их основные достоинства и недостатки.

В качестве основного средства для разработки под MeeGo предполагается использовать фреймворк *Qt*, о котором и пойдет речь далее. *Qt* представляет из себя мощный кроссплатформенный инструментарий, позволяющий разработчику унифицировано работать с различными целевыми платформами. На данный момент *Qt* поддерживает большое число модулей, позволяющих реализовывать различную функциональность программы. Кроме того, *Qt* отличается подробной и хорошо структурированной документацией, что существенно облегчает работу с ним.

Основным каналом распространения приложений для MeeGo является Интернет-магазин приложений AppUp. Об использовании этого ресурса и адаптации MeeGo-программ для него будет сказано в конце лекции.

3.2. Развертываниие MeeGo SDK

MeeGo SDK версии 1.0 — это образ файловой системы с развёрнутой в ней ОС MeeGo и предустановленными инструментами для разработки. Существует несколько вариантов развертывания MeeGo SDK.

Разработка непосредственно в MeeGo. Наиболее естественным вариантом разработки под MeeGo, казалось бы, должен быть именно этот способ. Он подразумевает установку ОС MeeGo на ту машину, на которой ведётся разработка. Все необходимые для разработки инструменты являются частью ОС MeeGo и могут быть установлены без каких-либо сложностей на рабочей машине. Рарзрабатываемое приложение в этом случае запускается и отлаживается локально, на той же машине, на которой ведётся разработка, естественным образом используя все библиотеки и подсистемы MeeGo. Использование этого метода гарантирует разработчику полное отсутствие проблем, связанных с механизмами виртуализации и эмуляции подсистем целевого устройства, а равно проблем, связанных с настройкой взаимодействия системы, на которой ведётся разработка и отладочной системой.

Однако многие разработчики найдут такой способ разработки неудобным, поскольку он подразумевает использование MeeGo в качестве основной ОС на рабочем компьютере, в то время, как пользовательский интерфейс MeeGo ориентирован прежде всего на портативные устройства — такие, как нетбуки, а не на desktop-компьютеры с большими мониторами. Очевидно также, что данный метод не может применяться при разработке приложений для смартфонов и других PDA, в силу ограниченности их пользовательского интерфейса.

MeeGo под QEMU. QEMU — это свободная программа с открытым исходным кодом для эмуляции аппаратного обеспечения различных платформ. QEMU предоставляет набор необходимых аппаратных компонент для установки гостевой операционной системы, в

качестве которой и устанавливается MeeGo. Для разработки под MeeGo используется модификация QEMU: QEMU-GL, которая позволяет гостевой системе использовать графический ускоритель host-системы.

Метод разработки ПО под MeeGo с использованием QEMU имеет довольно низкую производительность, связанную с необходимостью интерпретации машинных команд эмулятором процессора. Кроме того, для использования этого метода требуются процессор с поддержкой виртуализации и графический ускоритель.

MeeGo в chroot среде. Chroot — это операция изменения корневого каталога в Unix-подобных операционных системах. Программа, запущенная с использованием команды chroot, имеет доступ к каталогам и файлам, находящимся лишь в том каталоге, который был указан при запуске. Это удобный способ выполнения программы в своеобразной «песочнице», в которой предустановлено все необходимое для разработки программное обеспечение. При этом все процессы запускаются на той машине, на которой ведется разработка, а инструменты для разработки также запускаются под командой chroot. Для работы графического UI MeeGo в окно на хостовой системе используется X-сервер Xephyr.

К недостаткам этого метода можно отнести требование графического ускорителя GPU (graphics processing unit) от Intel.

Кросс-компиляция и удаленная отладка на целевом устройстве. В этом случае разработка ведется под любой ОС Linux. Далее при помощи компилятора gcc производится кросс-компиляция, т.е. компиляция под целевую платформу. Затем следует выполнение программы на целевой архитектуре и удаленная отладка при помощи gdb. При этом отладка выполняется на настоящей архитектуре и в реальных, неэмулируемых условиях.

Недостатком этого подхода можно считать малое количество устройств, на которых на этом этапе можно запустить MeeGo. Из смартфонов для этой цели подходят только Nokia N900 и специальный прототип для разработчиков Aava.

3.3. Технология *Qt*

Qt – это инструментарий, включающий в себя программный фреймворк, библиотеку элементов графического интерфейса и набор программ для разработки, – который используется для разработки межплатформенных приложений с графическим пользовательским интерфейсом преимущественно на языке C++. Однако, в различное время были созданы интерфейсы, позволяющие вести разработку с использованием Qt и на других языках программирования, таких как: Python – PyQt, PySide; Ruby – QtRuby; Java – QtJambi; PHP – PHP-Qt и другие.

Инструментарий Qt лежит в основе популярной среди пользователей Unix-подобных систем среды рабочего стола KDE, а также таких приложений, как Skype, VLC, Virtual Box и многих других.

Использование API Qt вместо других, специфичных для платформы, программных интерфейсов, позволяет создавать приложения, которые, во многих случаях, без всяческих доработок будет компилироваться и исполняться на любой из ОС поддерживаемых *Qt*, а в большинстве других случаев требовать лишь незначительной доработки. Среди таких ОС, помимо MeeGo – Windows, Mac OS X, различные дистрибутивы Linux, Solaris, использующие оконную систему X11, Symbian, Windows CE.

3.3.1. Краткая история

Разработка *Qt* как графического toolkit (библиотеки графических компонентов) была начата в 1991 году Гаавардом Нордом и Айриком Шамбе-Ингом, основавшими впоследствии компанию Quasar Technologies, затем переименованную в Trolltech. Идея разработки кроссплатфоменного toolkit появилась во время работы над графическим приложением для медицинской индустрии, которое должно было работать в ОС Windows и Unix. Буква Q появилась в названии фреймворка, поскольку Гааварду очень нравилось её начертание в шрифте, использовавшемся в редакторе Emacs. Буква t, за которой скрывается слово «toolkit», была добавлена по аналогии с Xt – X Toolkit, библиотекой для создания виджетов в оконной системе X.

Несколько лет проект разрабатывался без представления на рынке. Первый релиз Qt был сделан в 1995 году. Он включил в себя набор графических компонент для X11/Unix и Windows. В версии 3.0, вышедшей в 2001 году, появилась также поддержка Mac OS X.
В различное время фреймворк Qt распространялся под разными лицензиями. Если версия Qt для оконной системы X11 изначально выпускалась как под коммерческой, так и под бесплатной (хотя и не свободной) лицензией с открытым исходным кодом, то первые версии для Windows и Mac OS X существовали лишь в версии для коммерческого использования. Особую остроту вопрос лицензирования технологии приобрел с ростом популярности оконной среды KDE среди пользователей Linux в конце 90-ых годов, когда стало очевидно, что одна из важнейших компонент наиболее популярной свободной ОС не является свободным ПО. Проблема лицензирования X11-версии была решена при помощи перехода на свободную лицензию QPL и основания KDE Free Qt Foundation — организации, гарантирующей, что в случае, если разработка свободной версии Qt будет приостановлено, последняя версия будет выпущена под лицензией типа BSD.

Хотя к 2003 году версии *Qt* для OS X и для X11 выпускались под свободными лицензиями, версия для Windows по-прежнему выпускалась лишь под коммерческой лицензией. Это привело к тому, что в 2002 группа независимых разработчиков начала работу по портированию X11-версии фреймворка, выпущенной под лицензией GPL, на Windows. Работа эта, впрочем, не была завершена, поскольку в 2005 была выпущена версия фреймворка 4.0, в действие лицензии GPL было распространено на версии для всех поддерживаемых платформ. Добавленное позднее специальное исключение в лицензию, сделало возможным использование GPL-версии *Qt* в проектах, использующих одну из целого ряда свободных лицензий, таких, как BSD License, Eclipse Public License и других.

В 2008 году компания Trolltech была приобретена компанией Nokia и переименована сперва в *Qt* Software, а впоследствии — в *Qt* Development Frameworks. Вскоре после этого была выпущена версия фреймворка для основной мобильной ОС, использующейся Nokia — Symbian S60. С развитием другой мобильной ОС,

разрабатываемой Nokia — Maemo, в *Qt* была добавлена поддержка и этой платформы.

В версии *Qt* 4.5, вышедшей 14 января 2009 г., в фреймворк была добавлена третья опция лицензирования — LGPL, что сделало возможным использование «бесплатной» версии *Qt* в проектах с закрытым кодом (при выполнении некоторых условий).

3.3.2. Преимущества использования *Qt*

Основным преимуществом программирования с *Qt* является в упрощении и унификации процесса разработки программного обеспечения для различных целевых платформ. Сложности, возникающие, при портировании проектов с одной платформы на другую, очевидны. В силу различия архитектур ОС и отсутствия общепринятых стандартов и интерфейсов, код приложения оказывается насквозь пронизанным обращениями к специфичными для платформы API. Это становится особенно заметным в участках кода, отвечающих за графический пользовательский интерфейс, однако зачастую даже безобидные с виду участки, использующие стандартизированные API, оказываются труднопортируемыми.

Qt в значительной степени облегчает решение этой проблемы, предоставляя широчайший набор унифицированных программных интерфейсов. Вместо API операционной системы разработчик использует API *Qt*. API *Qt* реализован для каждой конкретной целевой архитектуры и опирается на нативные API операционной системы. В силу этого, приложение, написанное с использованием API Qt, фактически использует высокую производительность нативных интерфейсов целевой платформы; часто библиотеки Qt являются лишь тонкой прослойкой между приложением и API ОС.

Ещё одно несомненное преимущество *Qt* состоит в том, что его API позволяет скрыть сложные интерфейсы внешних библиотек. Порой для выполнения некоторой достаточно простой операции с использованием API ОС, программисту приходится изучать объёмную документацию и реализовывать тяжеловесные функции, инициализирующие и деинициализирующие применяемую библиотеку и т. д. Наглядным примером этой проблемы является создание оконных приложений с использованием Windows API.

Другой аспект использования *Qt* заключается в унификации кода приложения. Крупный проект зачастую использует значительное число внешних библиотек, многие из которых используют весьма специфичные по стилю интерфейсы, что порождает разнородные участки кода. API *Qt* используют единый стиль и подход, что позволяет сделать ваш код более легко читаемым и ясным.

3.3.3. Основные библиотеки фреймворка *Qt*

Итак, *Qt* позволяет запускать написанное с его помощью ПО в большинстве современных операционных систем путём простой компиляции программы для каждой ОС без изменения исходного кода. Он включает в себя основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML.

Кратко опишем ключевые библиотеки, входящие в дистрибутив:

- QtCore – базовые примитивы, не имеющие отношения к GUI;
- QtGui – примитивы GUI Phonon и QtMultimedia – библиотеки для работы с мультимедиа;
- QtNetwork – работа с сетью;
- QtOpenGL – поддержка OpenGL-графики;
- QtXml – работа с XML;
- QtSql – работа с SQL-базами данных;
- QtScript – позволяет использовать скриптовый язык, аналогичный JavaScript в Qt-приложениях;
- QtWebKit – позволяет работать с веб-движком (библиоткой для обработки и отображения Web-страниц) WebKit.

Для разработки кросс-платформенных приложений для мобильных устройств компания *Qt* Software разработала

дополниельную библиотеку Qt Mobility, пока не входящую в состав основного дистрибутива. Qt Mobility должен обеспечить удобную разработку приложений для мобильных платформ, поддерживающих *Qt* и, в первую очередь, ОС MeeGo.

Qt Mobility предоставляет интерфейс для функциональностей, специфичных для мобильных устройств, таких как, например:

- сервисы (GSM-связь, Bluetooth) ;
- записная книжка;
- мгновенные сообщения;
- органайзер;
- устройства позиционирования;
- сенсоры (акселерометр, датчик освещённости).

3.3.4. Инструменты разработки на *Qt*

В пакете *Qt* SDK поставляется набор инструментов, которые облегчают разработку приложений с использованием фреймворка. Перечислим основные:

- QtCreator – кроссплатформенная IDE для работы с фреймворком *Qt*, разработанная *Qt* Software. Эта IDE была специально разработана для работы с *Qt*, имеет возможности удаленной отладки, расширения плагинами, встроенный QtDesigner и QtAssistant и графический фронтенд для gdb. QtCreator входит в состав SDK.
- QtDesigner – инструмент для визуального дизайна графических интерфейсов. В результате работы Qt Designer создается xml файл, описывающий графический интерфейс.
- QtLinguist – локализация интерфейса.
- QtAssistant – система справки.
- QtSimulator – эмулятор мобильных устройств.

- qmake – система сборки.
- moc – метаобъектный компилятор , предварительная система обработки исходного кода. Позволяет использовать механизм слотов и сигналов. Утилита moc ищет в заголовочных файлах на C++ описания классов, содержащие макрос Q_OBJECT, и создаёт дополнительный исходный файл на C++, содержащий реализацию дополнительных методов.
- uic – компилятор графических интерфейсов, который получает на вход xml файл, сгенерированный QtDesigner, и по нему выдает код на C++.
- rcc – компилятор ресурсов.

3.3.5. Система сборки qmake

qmake – программное средство, с помощью которого упрощается процесс сборки проекта при разработке для разных платформ. qmake автоматизирует создание файла сборки Makefile, используя для этого более простой и лаконичный файл *.pro.

Утилита создает Makefile, основываясь на информации в файле проекта. Файлы проекта обычно создаются разработчиком, однако для их первичного создания можно также использовать и саму утитлиту qmake, запуская её с аргументом -project. qmake содержит дополнительные возможности для поддержки разработки с *Qt*, включая автоматическое создание правил для moc и uic.

Рассмотрим простой пример работы с qmake. Допустим, что у вас уже завершена начальная реализация вашего приложения, и у вас имеются следующие файлы: hello.cpp, hello.h, main.cpp. Используя текстовый редактор, создайте файл с названием hello.pro. Теперь в этот файл следует добавить строки, которые сообщают qmake об исходных файлах, файлах-заголовках,

используемых библиотеках, которые следует прилинковать в процессе сборки и др.

На первом шаге добавим исходные файлы в файл проекта. Чтобы это сделать, нужно использовать переменную SOURCES. Надо написать новую строку с SOURCES += и добавить hello.cpp после нее. Должно получиться наподобие:

```
SOURCES += hello.cpp
```

Теперь нужно повторить эти действия для каждого исходного файла в проекте. В итоге в нашем примере получается следующее:

```
SOURCES += hello.cpp
SOURCES += main.cpp
```

Или в одну строку:

```
SOURCES = hello.cpp \ main.cpp
```

Кроме исходных файлов должны быть указаны файлы заголовка. Для их добавления используется переменная HEADERS. После изменений наш файл выглядит так:

```
HEADERS += hello.h
SOURCES += hello.cpp \ main.cpp
```

Имя файла результата сборки устанавливается автоматически; оно такое же, как и имя файла проекта, но с суффиксом, соответствующим платформе. Например, если файл проекта называется - hello.pro, результатом сборки будет файл hello.exe для Windows и hello для Unix. Другое имя файла для результата сборки может быть указано в переменной target. Например,

```
TARGET = helloworld
```

Далее установим переменную CONFIG, отвечающую за общую конфигурацию сборки. Так как наш приложение использует Qt, то нужно поместить qt в строке CONFIG для того, чтобы qmake добавил релевантные библиотеки и обеспечил встроенные строки для moc и uic, включаемые в создаваемый файл сборки. Если в переменной CONFIG указать значение debug, то будет создана отладочная версия программы.

Переменная QT позволяет указать, какие модули Qt использует.

Для тестового файла укажем, что мы хотим использовать библиотеки ядра (core), XML (xml) и библиотеки работы с сетью (net). В результате в файл будет записана следующая строка:

```
QT += core xml network
```

Переменная LIBS перечисляет внешние библиотеки, которые мы хотим прилинковать к приложению, в виде ключей для линковщика. В примере прилинкуем к приложению библиотеку ncurses:

```
LIBS += -lncurses
```

Зачастую возникает необходимость собирать приложение в разных вариантах, например, для разных целевых платформ. В qmake для поддержки различных видов сборки существует механизм scopes, который позволяет создавать условные блоки и в зависимости от выполнения условий переходить в те или иные состояния. Добавим небольшой пример и в наш файл. Введем условные блоки, которые в зависимости от целевой платформы будут добавлять исходные файлы в проект. Так, например для windows будет добавлен файл hellowin.cpp :

```
win32 {
    SOURCES += hellowin.cpp
}
```

А для unix hellounix.cpp:

```
unix {
    SOURCES += hellounix.cpp
}
```

Также при помощи простой функции exists проверим, существует ли файл main.cpp:

```
!exists( main.cpp ) {
    error( "No main.cpp file found" )
}
```

Запишем полностью получившийся qmake файл:

```
CONFIG += qt
QT += core xml network
HEADERS += hello.h
SOURCES += hello.cpp
```

```
SOURCES += main.cpp
LIBS += -lncurses
win32 {
    SOURCES += hellowin.cpp
}
unix {
    SOURCES += hellounix.cpp
}
!exists( main.cpp ) {
    error( "No main.cpp file found" )
}
```

Теперь qmake можно использовать для создания файла сборки приложения. В командной строке в каталоге с проектом нужно написать:

```
qmake hello.pro
```

Затем может быть запущена утилита make или nmake для сборки проекта.

3.3.6. Механизм сигналов и слотов

Сигналы и слоты используются для обмена сообщениями между объектами. Механизм сигналов и слотов является особенностью *Qt*. Необходимость в подобном механизме возникает, когда требуется, чтобы при изменении одного объекта, оповещался другой. Так, например, при разработке графического интерфейса, при нажатии на кнопку «Закрыть» вызывается метод окна close().

Техника сигналов и слотов реализована следующим образом: сигнал вырабатывается, когда происходит определенное событие, а слот — это функция, которая вызывается в ответ на определенный сигнал. Каждый класс может объявлять сигналы, которые он будет отправлять и слоты, которые можно ассоциировать с конкретными сигналами. При этом сигналы и слоты слабо связаны. Класс, который вырабатывает сигнал, не знает и не заботится о том, какие слоты его получат.

Сигналы и слоты могут иметь аргументы. Механизм сигналов и слотов Qt гарантирует, что если мы подключим сигнал к слоту, слот будет вызван с параметрами сигнала в нужное время.

Система слотов и сигналов реализована как надстройка над синтаксисом C++. Исходный файл обрабатывается метакомпилятором moc, который генерирует вспомогательные файлы. При этом ограничения метакомпилятора накладывают определенные ограничения на классы, использующие слоты и сигналы для взаимодействия. Так, например, такие классы не могут использовать механизм шаблонов C++.

Рассмотрим небольшой пример использования механизма слотов и сигналов (см. рис. 3.1).

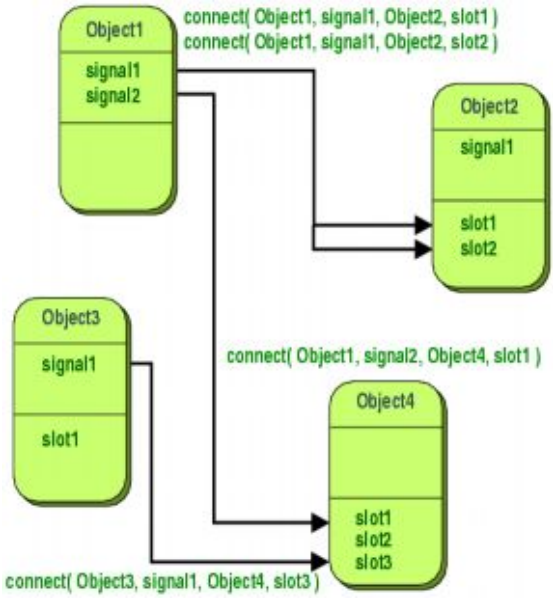


Рис. 3.1.

Класс, наследуемый от QObject будет выглядеть следующим образом:

```
#include <QObject>
class Counter : public QObject
{
    Q_OBJECT
public:
```

```
    Counter() { m_value = 0; }
    int value() const { return m_value; }
public slots:
    void setValue(int value);
signals:
    void valueChanged(int newValue);
private:
    int m_value;
};
```

Класс, наследованный от QObject, имеет то же самое внутреннее состояние и обеспечивает публичные методы для доступа к этому состоянию, но дополнительно у него есть поддержка для использования сигналов и слотов. Этот класс может сообщить внешнему миру, что его состояние изменилось, выработав сигнал valueChanged() и у него есть слот, в который другие объекты могут посылать сигналы.

Все классы, содержащие сигналы и слоты должны указывать макрос Q_OBJECT в начале их описания. Они также должны быть потомками (прямо или косвенно) QObject.

Слоты реализуются программистом. Возможная реализация слота Counter::setValue() выглядит следующим образом:

```
void Counter::setValue(int value)
{
    if (value != m_value) {
        m_value = value;
        emit valueChanged(value);
    }
}
```

Ключевое слово emit вырабатывает сигнал valueChanged() объекта с новым значением в качестве аргумента.

В следующем примере мы создаем два объекта типа Counter и соединяем сигнал valueChanged() первого со слотом setValue() второго используя статическую функцию QObject::connect():

```
Counter a, b;
```

```
QObject::connect(&a, SIGNAL(valueChanged(int)),  &b,
SLOT(setValue(int)));
a.setValue(12);// a.value() == 12, b.value() == 12
b.setValue(48);// a.value() == 12, b.value() == 48
```

Вызов a.setValue(12) вырабатывает сигнал valueChanged(12), который получит объект b в свой слот setValue() slot, т.е. будет вызвана функция b.setValue(12). Тогда b вырабатывает такой же сигнал valueChanged(), но так как он не подключен ни к одному слоту, это сигнал будет проигнорирован.

Отметим, что функция setValue() устанавливает новое значение и вырабатывает сигнал только есть value != m_value. Это предотвращает бесконечный цикл в случае кругового соединения (например, если бы b.valueChanged() был бы подключен к a.setValue()).

Сигнал вырабатывается для каждого соединения. Если соединение продублировать, будут выработаны два сигнала. Соединение всегда можно разорвать, использовав функцию QObject::disconnect().

Приведенный выше пример показывает, как объекты могут работать вместе без необходимости знать что-либо друг о друге.

3.4. Магазин приложений AppUp

AppUp – магазин приложений для Windows и Moblin, в первой половине 2011 года был адоптирован для распространения приложений MeeGo. Чтобы пользователь мог скачивать приложения в AppUp, ему необходимо установить на свой компьютер MeeGo AppUp Client, в мае стала доступна его beta-версия. Несмотря на то, что Moblin является предком MeeGo и давно поддерживается в AppUp, Moblin AppUp Client в MeeGo использовать нельзя.

Программисту для создания программ, совместимых с AppUp, необходимо скачать Intel AppUp SDK для MeeGo. SDK для Moblin

также не подходит. Для работы в QtCreator существует соответствующий плагин.

Типичный пример интеграции в приложение выглядит следующим образом:

```
#include "adpcore.h"
int main( int argc, char* argv[] ) {
    ADP_RET_CODE ret_code;
    ADP_APPLICATIONID GUID =
{{0x0,0x11111111,0x11111111,0x11111111}};
    if ((ret_code = ADP_Initialize()) != ADP_SUCCESS
){
        printf( "ERROR: exiting" );
        exit( -1);
    }
    if (( ret_code = ADP_IsAuthorized(GUID)) ==
ADP_AUTHORIZED )
        printf( "Hello World" );
    else
        printf( "Not authorized to run" );
}
```

Заголовочный файл adpcore.h содержит информацию об API AppUp, в частности, описывает типы ADP_RET_CODE и ADP_APPLICATIONID. Переменная GUID содержит идентификатор приложения. Функция ADP_IsAuthorized() по GUID приложения проверяет, авторизован ли пользователь на его запуск.

Уникальный GUID выдается сервисом AppUp для каждого приложения, которое будет там размещено, его можно получить, заполнив специальную форму. Также существуют зарезервированные GUID: ADP_DEBUG_APPLICATIONID – для запуска приложений в режиме отладки, когда проверка прав пользователя нежелательна, ADP_EXPIRED_APPLICATIONID – для проверки поведения приложения в случае, если срок лицензии на него истек.

Кроме того, с помощью функций ADP_ApplicationBeginEvent() и ADP_ApplicationEndEvent() программист может собирать статистику использования своего приложения клиентами AppUp. Также у программиста есть возможность собрать статистику критических сбоев с помощью ADP_ReportCrash().

3.5. Выводы

В этой лекции мы провели обзор MeeGo SDK и различных вариантов развертки, которые он предоставляет. Мы познакомились с входящим в дистибутив инструментарием *Qt* — основным средством разработки для MeeGo. *Qt* позволяет разрабатывать широкий спектр приложений, используя стнадартный API и переносить их на любую целевую платформу, поддерживаемую *Qt*, с минимальными изменениями исходного кода. Магазин приложений Intel AppUp является основным каналом распространения приложений для MeeGo.

3.6. Контрольные вопросы

1) Как правильно расшифровывать аббревиатуру SDK в материале этой лекции?
    1. System Design Kit
    2. Software Development Kit
    3. Self-Development Kit
    4. Skin Decontamination Kit
2) Какой вариант развёртки не предлагается для MeeGo SDK 1.0?
    1. исполнение MeeGo на эмуляторе QEMU
    2. исполнение MeeGo на целевом устройстве
    3. запуск MeeGo на целевом устройстве с использование команды chroot
    4. запуск MeeGo под виртуальной машине VMWare
3) Что выполняет команда chroot?

1. запуск приложения с указанием произвольного каталога в качестве корневого
2. запуск приложения от имени иного пользователя, чем текущий
3. переход в указанную директорию
4. переход в корневую директорию

4) Что можно отнести к недостаткам разработки для MeeGo на целевой платформе?
   1. пользовательский интерфейс MeeGo не ориентирован на разработчика
   2. в MeeGo не могут быть установлены все необходимые инструменты для разработки
   3. сложности с виртуализацией
   4. варианты 1 и 2 верны
5) Что такое QEMU?
   1. оптимизированный компилятор для языка C++
   2. программа для эмуляции аппаратного обеспечения различных платформ
   3. графический ускоритель
   4. Набор стандартов, определяющих базовый API для создания приложений переносимых на различные платформы
6) Каково назначение модуля GL, который входит в стандартную сборку QEMU для MeeGo?
   1. эмуляция процессоров семейства Intel Atom
   2. использование аппаратного графического ускорителя для более быстрой отрисовки графики
   3. ускорение исполнения программ за счёт выполнения части кода на реальном процессоре, без эмуляции
   4. в QEMU такого модуля нет
7) Какой инструмент используется для управления конфигурациями MeeGo SDK?
   1. chroot
   2. MADDE
   3. Xephyr
   4. sudo
8) Укажите, что из перечисленного не является обязательным требованием к системе, на которой выполняется QEMU,

при запуске MeeGo в среде QEMU?

1. процессор с поддержкой виртуализации
2. графический ускоритель
3. установленная ОС Linux
4. установленный Qt Creator

9) Какую основную задачу решает команда chroot?
   1. выполнение процесса в «песочнице», в изолированной среде
   2. выполнение процесса с правами пользователя root
   3. выполнение процесса на удаленной машине
   4. обеспечение безопасности в ОС Linux
10) Приложения, запускаемые в chroot-среде, исполняются:
   1. на удаленной машине, доступ к которой осуществляется по ssh
   2. на той же машине, на которой ведется разработка
   3. на виртуальной машине VirtualBox
   4. в среде эмулятора QEMU
11) Какой инструмент используется для отображения графической среды MeeGo в окне хостовой системы при разработке под chroot?
   1. X-сервер Xephyr
   2. оконный менеджер mutter
   3. QEMU
   4. Виртуальная машина VirtualBox
12) Что можно отнести к недостаткам разработки для MeeGo в chroot-среде?
   1. замедленная работа приложений
   2. обязательное наличие GPU от Intel
   3. не могут быть установлены все необходимые инструменты для разработки
   4. варианты 2 и 3 верны
13) Что такое кросс-компилятор?
   1. компилятор, который запускается на удалённой машине

2. компилятор, производящий исполняемый код для платформы, отличной от той, на которой исполняется сам
3. компилятор, производящий исполняемый код для исполнения в виртуальной машине VirtualBox
4. этим термином называется любой компилятор, производящий код для ОС MeeGo

14) Что представляет собой MeeGo SDK версии 1.0?
    1. кросс-компилятор
    2. кросс-компилятор, набор заголовочных файлов и двоичных файлов библиотек
    3. образ диска виртуальной машины VirtualBox
    4. raw-образ ФС с установленной ОС MeeGo
15) Какой инструмент позволяет выполнять удаленную отладку на целевом устройстве
    1. gcc
    2. gdb
    3. strace
    4. Qt
16) Какие преимущества есть у метода отладки на целевом устройстве?
    1. отладка выполняется «по-живому» и возникает ясное представление о выполнении программы на целевом устройстве, ее производительности и возможных проблемах
    2. этот метод наиболее прост в настройке
    3. при использовании этого метода отлаживаемое приложение запускается наиболее быстро.
    4. у метода отладки на целевом устройстве нет особых преимуществ и его не рекомендуется использовать
17) Что можно назвать основным недостатком метода отладки на целевом устройстве?
    1. малое количество устройств, на которых может быть запущен handset-вариант MeeGo на сегодняшний

2. отладка на целевом устройстве требует наличия в системе графического ускорителя Intel
3. снижение скорости выполнения программ

18) На каких мобильных устройствах устройстве на сегодняшний день может быть запущен handset-вариант MeeGo
    1. Aava Phone
    2. iPhone 3G
19) Что означает аббревиатура Qt?
    1. Quantum Topology
    2. Qualification Test
    3. аббревиатура ничего не означает, просто буква Q имела красивое начертание в том шрифте, который один из авторов использовал в своем Emacs, а t была добавлена по аналогии с инструментом Xt.
20) Что нельзя отнести к основным преимуществам использования Qt?
    1. обёртка с простым интерфейсом для порой очень сложных API
    2. компиляция в промежуточный код, которая позволяет переносить его на любую платформу
    3. единый интерфейс для всех платформ
    4. использование мощности native-интерфейсов целевой платформы
    5. переносимость кода
21) Когда и кем была начата разработка фреймворка Qt?
    1. 1991, Haavard Nord и Eirik Chambe-Eng
    2. 1996, Haavard Nord и Eirik Chambe-Eng
    3. 2008, компания Trolltech
    4. 2008, корпорация Nokia
22) В каком году был выпущен первый релиз фреймворка Qt?
    1. 1996
    2. 1995
    3. 1994

4. 2008

23) Под какой лицензией была выпущена первая версия фреймворка Qt для Unix?
   1. GPL
   2. LGPL
   3. проприетарная лицензия

24) Под какой лицензией была выпущена первая версия фреймворка Qt для Windows?
   1. GPL
   2. LGPL
   3. проприетарная лицензия
   4. FreeQt

25) В какой версии Qt действие лицензии GPL было распространено на все платформы?
   1. Qt 4
   2. Qt 3.0
   3. Qt 2.1
   4. Qt изначально предлагался под лицензией GPL для всех поддерживаемых платформ

26) Какой компанией в 2009 году была приобретена компания Trolltech?
   1. Microsoft
   2. Intel
   3. Nokia
   4. Trolltech должен был быть приобретён Apple, но сделка в итоге сорвалась

27) Для какой версии фреймворка Qt была добавлена лицензия LGPL?
   1. Qt 3.0
   2. Qt 2.1
   3. Qt 4.5
   4. фреймворк Qt никогда не распространялся под лицензией LGPL

28) Какая из перечисленных библиотек фреймворка Qt

содержит базовые примитивы, не имеющие отношения к GUI?

1. QtNetwork
2. QtWebKit
3. QtCore
4. QtOpenGL

29) Какая из перечисленных библиотек фреймворка Qt обеспечивает работу с сетью?

1. QtNetwork
2. QtCore
3. QtOpenGL
4. QtScript

30) Как называется библиотека фреймворка Qt, которая предназначена для работы с SQL базами данных?

1. QtDB
2. QtSqlLib
3. QtSql
4. в Qt нет библиотеки для работы с базами данных

31) Какая из перечисленных библиотек фреймворка Qt позволяет работать с Web-движком?

1. QtNetwork
2. QtWebKit
3. QtCore
4. QtOpenGL
5. QtScript

32) Какая из перечисленных библиотек фреймворка Qt позволяет позволяет использовать скриптовый язык, аналогичный JavaScript в Qt-приложениях?

1. QtNetwork
2. QtWebKit
3. QtCore
4. QtOpenGL
5. QtScript

33) Какая из задач не решается библиотекой QtMobility?

1. взаимодействие с мобильными Java-приложениями
2. взаимодействие с системами обмена мгновенными сообщениями
3. взаимодействие с системами позиционирования
4. работа со списком контактов

34) Какую версию QtMobility поддерживает дистрибутив MeeGo 1.1?
1. версию 1.0
2. версию 1.0.2
3. версию 1.1
4. в настоящий момент поддержка QtMobility в MeeGo не реализована

35) Начиная с какой версии MeeGo имеет поддержку QtMobility?
1. с версии 1.1
2. с версии 1.1
3. с версии 0.5
4. MeeGo не имеет поддержки QtMobility

36) Что такое QtCreator?
1. эмулятор мобильных устройств
2. IDE для разработки на C++ и QML
3. инструмент для создания графических интерфейсов с использованием виджетов Qt
4. компилятор для приложений, написанных под фреймворком Qt

37) Какой из указанных инструментов занимается локализацией интерфейса?
1. QtDesigner
2. QtLinguist
3. QtAssistant
4. Qt Simulator

38) Какой из указанных инструментов используется для компиляции языка описания графических интерфейсов Qt в код на C++?

1. qmake
2. moc
3. uic
4. rcc

39) Какой из следующих инструментов обрабатывает файлы *.pro?

Варианты:

1. qmake
2. moc
3. uic
4. make

40) Что генерирует утилита qmake?

1. Makefile.am — входные данные для приложения automake
2. Makefile — входные данные для приложения make
3. shell-скрипт для сборки проекта
4. исполняемый файл

41) Каким образом реализован механизм слотов и сигналов?

1. при помощи механизма шаблонов C++
2. как надстройка над языком C++ вне стандарта
3. при помощи функций обратного вызова (callbacks)
4. варианты 1 и 3 верны

42) Каково назначение механизма слотов и сигналов?

1. проверка безопасности типов
2. безопасное обращение к памяти, выделенной под объекты
3. обмена сообщениями между объектами
4. обёртка над механизмом сигналов в Unix-подобных системах

43) Слоты могут объявляться?

1. как функции в любом месте кода
2. как функции-члены с модификатором доступа "protected" в любых классах C++

3. как функции-члены классов, наследующих QObject, в объявление которых включено макроопределение Q_OBJECT
4. только как функции-члены в классах, наследующих QWidget, в объявление которых включено макроопределение SLOT

44) Каким образом можно отправить сигнал?
1. при помощи функции connect
2. при помощи макроопределения emit
3. при помощи вызова функции send
4. в Qt не предусмотрена явная отправка сигналов

Список литературы

MeeGo wiki
1. http://wiki.meego.com/MeeGo_SDK_Development_Options
2. http://wiki.meego.com/Getting_started_with_the_MeeGo_SDK_for_Linux
3. Qt Documentation http://doc.qt.nokia.com/4.7/index.html.
4. Intel AppUp SDK Suite http://appdeveloper.intel.com/en-us/meego-sdk-suite

4. Разработка приложений для планшетных компьютеров

Планшетные компьютеры. Сенсорный экран. Датчики.

4.1. Введение

Последнее время наряду с ноутбуками и смартфонами набирает популярность еще одна разновидность мобильных компьютеров – планшетные. Наиболее известным примером такого компьютера является Apple iPad. От ноутбуков планшеты отличает отсутствие традиционной клавиатуры, от смартфонов – большие размеры. Они ориентированы в первую очередь на развлечение пользователя, пассивное восприятие информации. В линейке аппаратных платформ для ОС MeeGo планшеты занимают одно из главных мест.

В данной главе мы рассмотрим аппаратные особенности планшетов с точки зрения прикладного программиста. Во-первых, это сенсорный экран, который в планшете заменяет почти все традиционные устройства ввода, такие как «мышь» и клавиатура. В случае «мыши» эти замена почти эквивалента по природе самого сенсорного экрана – оба они являются pointing device, и мы сосредоточимся как раз на этом случае.

Во-вторых, планшет обладает набором датчиков, например, датчиком ориентации, освещения, акселерометром и т.п. Для поддержания такого разнородного набора датчиков требуется самостоятельный API, расположенный в библиотеке *Qt* Mobility. Датчики, в свою очередь, позволяют реализовать более удобный, более интуитивный интерфейс управления программой на планшете, который был бы невозможен на нетбуке.

Кроме аппаратных особенностей, у планшетов есть особенности пользовательского интерфейса, для которого и исчезновение клавиатуры, и изменение размера экрана критичны. Эти

особенности также являются очень существенными, но мы их касаться не будем.

4.2. Сенсорный экран (touchscreen)

Сенсорный экран – универсальное устройство ввода, которое заменило в планшете и клавиатуру, и «мышь». По принципу действия он очень схож с мышью – пользователь прикосновением пальца к экрану указывает на нем точку и задает вид действия. Для ввода текста применяются разного рода экранные клавиатуры, которые являются самостоятельными программными решениями, мы не будем их рассматривать.

Сенсорный экран может быть использован различными способами. Самый простой и частый способ – использовать стандартные виджеты *Qt*, которые уже адаптированы для работы с сенсорным экраном и программисту не надо ничего специально делать для этого. Этот метод плохо подходит в ситуации, когда пользовательский интерфейс сложный или нестандартный – например, в играх. В этом случае целесообразно перехватывать системные события *Qt*, относящиеся к сенсорному экрану, и самостоятельно их обрабатывать.

Эти события бывают двух видов – события собственно от сенсорного экрана и события gestures. Gesture (буквально «жест») – это комбинация действий с сенсорным экраном, которые обозначают определенное событие для пользовательского интерфейса. Например, если человек установил два пальца на мультисенсорный экран и развел их в стороны – это соответствует gesture для увеличения масштаба содержимого виджета. Gesture могут быть как встроенные в систему, так и задаваемые программистом. Встроенные gesture служат для унификации опыта пользователя при работе с сенсорным экраном.

На более низком уровне абстрагирования находятся события непосредственно от сенсорного экрана, содержащие информацию о прикосновениях к нему. В *Qt* эти события и gesture могут появляться одновременно. Обработка этих событий позволяет получить наиболее полную информацию от сенсорного экрана в рамках его штатных возможностей.

Наконец, в исключительных случаях возможно работа с сенсорным экраном на уровне его драйвера в ядре MeeGo. Это наиболее сложный и трудоемкий способ и он имеет смысл только в том случае, если у конкретной модели сенсорного экрана есть какие-то необходимые нам возможности, которые не поддерживаются *Qt*. Необходимо также учитывать проблемы с переносимостью созданной таким образом программы.

Далее мы будем говорить о событиях сенсорного экрана. Для перехвата событий необходимо применить традиционную для Qt схему – создать свой собственный виджет, унаследовав его от какого-нибудь стандартного, и в нем переопределить метод event():

```
bool MyWidget::event(QEvent *event) {
    if (event->type() == QEvent::SomeEvent) {
        QSomeEvent *e =
static_cast<QSomeEvent*>(event);
           // ...
        e->accept();
        return true;
    }
    return BaseWidget::event(event);
}
```

В этом фрагменте кода важны четыре момента. Во-первых, нужно выделить только те события, которые нам интересны, не нужно пытаться обработать все события, которые получает виджет. Для этого в начале мы проверяем тип события, в данном случае QEvent::SomeEvent.

Во-вторых, сам по себе базовый класс событий QEvent малоинформативен, после того, как мы определились с типом события, надо привести его к соответствующему типу (в примере QSomeEvent). В *Qt* для этого рекомендуется использовать static_cast.

В-третьих, после обработки события его следует пометить как обработанное с помощью метода accept() и возврата true.

В-четвертых, все события, которые не были обработаны в новом обработчике, должны быть переданы обработчику базового класса.

Для сенсорного экрана выделены следующие типы событий: QEvent::TouchBegin (начало прикосновения), QEvent::TouchUpdate (продолжение прикосновения), QEvent::TouchEnd (завершение прикосновения). Класс QTouchEvent появился недавно, в *Qt* 4.6, для поддержки сенсорных экранов и сенсорных площадок (touchpad). Вложенный класс QTouchEvent::TouchPoint описывает точку прикосновения к сенсорному экрану. В QTouchEvent имеется список таких точек – например, если пользователь касается экрана четырьмя пальцами, то в списке будут четыре точки. В простейшем случае достаточно рассматривать первую точку списка. Кроме координат, с каждой точкой ассоциировано ее состояние, для простейшего анализа оно также не важно.

Таким образом, можно выстроить несколько стратегий обработки событий сенсорного экрана. В простейшем случае (клик) достаточно реагировать только на TouchBegin или TouchEnd, в более сложных случаях необходимо анализировать данные в TouchUpdat*e* на лету или накапливать их для последующего анализа при завершении прикосновения.

По умолчанию виджет не получает события от сенсорного экрана. Чтобы начать их получать, необходимо выполнить вызов setAcceptTouchEvents(true). Другая проблема, которая может возникнуть с сенсорным экраном – обработка событий gesture других виджетов может приводить к нежелательным эффектам. В этом случае необходимо также начать получать события желаемых gestures в своем виджете, выполнив, например, grabGesture(Qt::PanGesture), после чего в обработчике событий отмечать эти события как обработанные. Другие виджеты перестанут получать события и нежелательные эффекты прекратятся.

4.3. Датчики

Датчики – неотъемлемая часть современного планшета или смартфона. В продвинутых телефонах они появились очень давно – например, датчик, позволяющий телефону определить, приложен ли он к уху или должен работать в режиме громкой связи. Сейчас количество датчиков и их возможности существенно возросли. Это касается как датчиков высокого уровня (получение текущей ориентации экрана (портрет, пейзаж)), так и низкого уровня, как, например, получение в режиме реального времени показаний акселерометра.

Вместе с тем набор датчиков меняется от устройства к устройству, более того, трудно предугадать, какие новые датчики появятся в будущем. Данные, получаемые от датчиков, неоднородны. Эти ограничения делают API датчиков относительно сложным.

Поддержка датчиков в *Qt* расположена в компоненте Sensor библиотеки *Qt* Mobility. Основным классом компоненты является QSensor – базовый класс для различных датчиков. Класс QSensorReading предназначен для передачи данных от датчика в программу, от него также наследуются классы с данными для различных датчиков. Эти классы предназначены для прикладных программистов, которые хотят просто использовать датчики.

Для системных программистов, которые создают новые датчики, предназначены механизмы класса QSensorBackend. Создавая наследников этого класса, программист может подключаться к аппаратному датчику или системному сервису для получения от них данных нового датчика. Мы не будем рассматривать этот вопрос.

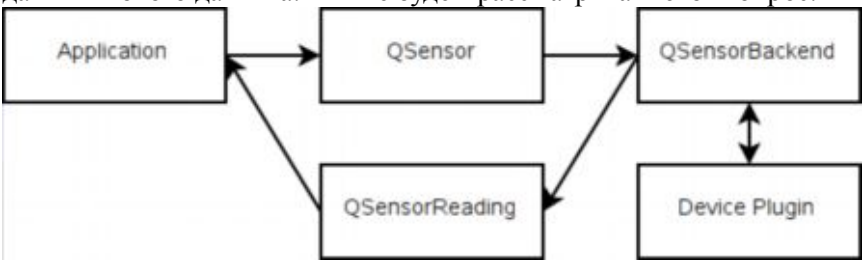


Рис. 4.1.

Qt знает множество типов датчиков:

- акселерометр (QAccelerometer)
- магнитометр (QMagnetometer)
- компас (QCompass)
- датчик освещённости (QAmbientLightSensor)
- датчик ориентации (QOrientationSensor)
- датчик поворота (QRotationSensor)
- датчик близости (QProximitySensor)
- и др.

Каждому типу соответствует одноименный класс – наследник QSensor. Получить типы всех датчиков, имеющихся в системе, можно с помощью статического метода:

```
QList<QByteArray> types =  QSensor::sensorTypes ();
```

Кроме типа, каждый датчик имеет уникальный строковый идентификатор, например, «n900.accelerometer». Получить идентификаторы всех датчиков заданного типа возможно, используя следующую конструкцию:

```
QList<QByteArray> sensor_ids =
QSensor::sensorsForType(types.last());
```

Зная тип и идентификатор датчика, можно начать с ним работу. Рассмотрим пример акселерометра:

```
QAccelerometer sensor(sensor_id);
sensor.start();
// Здесь получение и обработка данных
sensor.stop();
```

Отметим, что ни создание объекта sensor, ни вызов его метода start() не обязаны как-то инициализировать датчик, равно как и вызов метода stop() не обязан датчик останавливать. Вполне возможно, что тот же датчик давно используется каким-то другим приложением или самой ОС. Работая с классом QSensor, программист только добавляет нового «слушателя» для данных

датчика, *Qt* Mobility же координирует взаимодействие всех «слушателей» и аппаратуры.

Важный вопрос, который актуален для датчика – способ получения данных от него. Обычно выделяют два подхода – поллинг и прерывания. Поллинг – это периодический опрос датчиков с целью получить новые данные, реализуется проще всего и наименее вычислительно эффективно. В случае использования прерываний работа основной программы прерывается в момент, когда датчик передает новое значение. Технически это может быть реализовано по-разному – от аппаратных прерываний в микроконтроллерах до специальных событий в современных event-driven системах. В *Qt* естественно использовать механизм слотов и сигналов:

```
QObject::connect(sensor, SIGNAL(readingChanged()),
widget, SLOT(handleSensor()));
```

Отметим, что сам сигнал не несет никакой информации о новом значении датчика, он лишь информирует обработчик о факте изменения значения. Программист должен получить и обработать новое значение в handleSensor() явным образом, например, так:

```
QAccelerometerReading reading = sensor.reading();
qreal x = reading->x();
```

Таким образом, если отказаться от сигнала readingChanged() и просто вызывать handleSensor() раз в секунду (например, с помощью таймера), мы получим поллинг.

Некоторые датчики, например, акселерометр, меняют свои показания настолько часто, что механизм слотов и сигналов может быть перегружен потоком вызовов. Чтобы решить эту проблему, API датчиков включает в себя систему фильтров. Фильтр – это наследник класса QSensorFilter, в котором программист переопределяет один метод:

```
bool QSensorFilter::filter(QSensorReading* reading);
```

Этот метод должен возвращать истину в случае, если новое значение датчика должно активировать сигнал readingChanged() и ложь, если новое значение должно быть проигнорировано. Фильтр устанавливается на датчик с помощью метода

```
void QSensor::addFilter(QSensorFilter* filter);
```

Может быть установлено несколько фильтров, в этом случае сигнал будет активирован, только если все фильтры признали его годным.

Чтобы быстро познакомиться с датчиками, доступными ОС MeeGo на вашем устройстве, лучше всего использовать утилиту “Sensor Explorer”. Она входит в пакет qt-mobility-examples и располагается в /usr/lib/qtmobility/examples/sensor_explorer.

4.4. Выводы

В этой лекции были рассмотрены две основных аппаратных особенности планшетных компьютеров – сенсорный экран как универсальное устройство ввода и набор датчиков. Обе особенности отражают тот факт, что планшет – устройство в первую очередь для развлечения, а не для сложной работы.

Аппаратные особенности рассмотрены с точки зрения их API в *Qt*, обсуждены типовые способы их использования. Из интересных смежных тем, не отраженных в этом тексте, можно выделить использование и создание gestures, разного рода экранные клавиатуры и их заменители, пользовательские интерфейсы, ориентированные на сенсорные экраны, а также API для подключения новых датчиков.

4.5. Контрольные вопросы

1) В чем основное отличие планшета от нетбука ?
   1. Наличие предустановленной ОС
   2. Отсутствие клавиатуры
   3. Способность работать под управлением MeeGo

2) Что является основным устройством ввода планшета ?
   1. Сенсорный экран
   2. Клавиатура
   3. «Мышь»
   4. Акселерометр

3) Какой способ ввода текста на планшете является штатным?
   1. С помощью подключаемой аппаратной клавиатуры
   2. С помощью экранной клавиатуры
   3. С помощью Интернета

4) Поддерживают ли стандартные виджеты Qt и MeeGoTouch работу с сенсорным экраном ?
   1. Да
   2. Частично, MeeGoTouch поддерживает, Qt нет
   3. Нет, поддержку надо реализовать самостоятельно

5) В каком случае при переопределении метода event() надо вызывать метод базового класса ?
   1. Всегда
   2. Никогда
   3. В случае, если событие не обработано

6) Какое из перечисленных имен обозначает идентификатор типа события ?
   1. Qt::TouchPointMoved
   2. QTouchEvent::TouchPoint
   3. QEvent::TouchUpdate

7) Какие события обязательно нужно обрабатывать при работе с сенсорным экраном ?
   1. Достаточно обрабатывать какое-нибудь одно по выбору программиста
   2. TouchBegin и TouchEnd
   3. TouchBegin, TouchUpdate и TouchEnd

8) Что не важно с точки зрения проектирования API датчиков?
   1. Датчики измеряют и передают разнородные параметры
   2. Датчики размещены на мобильном устройстве

3. Набор датчиков на разных устройствах различен

9) Для чего предназначен класс QSensorReading ?
   1. Передача данных от датчика приложению
   2. Создание новых датчиков
   3. Получение списка датчиков в системе

10) В каком случае нужна фильтрация данных датчика ?
   1. В данных присутствуют помехи
   2. Для взаимной корректировки дублирующихся датчиков
   3. Датчик посылает данные слишком интенсивно

Список литературы

1. Using Multi-Touch and Gestures with Qt. http://www.slideshare.net/qtbynokia/using-multitouch-and-gestures-with-qt
2. QTouchEvent Class Reference. http://doc.qt.nokia.com/latest/qtouchevent.html
3. QtMobility 1.2 Sensors. http://doc.qt.nokia.com/qt-mobility-snapshot/sensors-api.html

5. Введение в психологию человеко-компьютерного взаимодействия

Эволюция подходов к проектированию человеко-компьютерного взаимодействия. Модели интерфейса. Специфика проектирования для мобильных устройств

5.1. Введение

Одним из ключевых факторов успеха программного обеспечения в современном мире является его удобство для пользователя. Парадигма создания ПО в последнее время претерпела значительные изменения. Все больше времени и ресурсов вкладывается в создание «дружелюбного» интерфейса, который позволит пользователю выполнять поставленные перед ним задачи быстро, легко, и не замечая технических аспектов работы системы.

Эта лекция является вводной и содержит основы знаний по теме проектирования человеко-компьютерного взаимодействия.

Используемые сокращения
ПИ — пользовательский интерфейс
ЧКВ — человеко-компьютерное взаимодействие

5.2. Эволюция подходов к проектированию человеко-компьютерного взаимодействия

Подходы к проектированию ПО, миновав ряд этапов, развивались эволюционно.

Машиноцентрический подход. Был общепринятым на заре появления технических систем.

Пользователями являлись обученные специалисты-программисты. Человек в данной системе рассматривался как ее звено, решающее различные задачи.

Антропоцентрический подход. Суть подхода заключалась в том, что машина является орудием труда, и ключевым в проектировании подобных систем является анализ деятельности оператора.
Однако подход был излишне «психологизирован». Ключевая роль отдавалась инженерным психологам, которые будучи специалистами в своей области не являлись таковыми в сфере технологий.

Системно-технический подход. Появился практически одновременно с антропоцентрическим. Роли человека и машины в нем уравнивались. Подход практически не получил развития, так как инженеры, которые играли здесь ведущую роль, не были специалистами в психологии и зачастую игнорировали психологическое знание.

Человеко-ориентированный подход. Явился менее радикальной формой антропоцентрического подхода, он постулирует что потребности (а также цели, возможности и пр.) человека необходимо учитывать, особенно на первых этапах проектирования нового продукта.

5.3. Дисциплины и подходы, в рамках которых разрабатываются методы и методики, использующиеся в проектировании ПИ

Определений следующих дисциплин великое множество. Здесь определения были выбраны таким образом, чтобы максимально разграничить упоминаемые дисциплины, так как их разделение иногда вызывает сложности даже у специалиста.

5.3.1. Академические дисциплины и подходы

Инженерная психология (Engineering psychology) — изучает психологические закономерности трудовой деятельности человека в системах управления и контроля, его информационное взаимодействие с техническими устройствами этих систем.

Эргономика (Human Factors) — научно-прикладная дисциплина, занимающаяся изучением и созданием эффективных систем, управляемых человеком.

Поскольку западные и отечественные подходы развивались изолированно, есть лишь примерное соответствие иностранных названий принятым в России терминам.

Человеко-компьютерное взаимодействие (Human-computer interaction) — изучает то, какими способами пользователи работают с компьютерами, и каким образом должны быть спроектированы компьютеры (и программная и аппаратная части), для того, чтобы они использовались с максимальной эффективностью.

Таким образом в процессе взаимодействия человека с компьютером инженерная психология более ориентируется на человека, изучение его особенностей и потребностей, а эргономика на учет этих факторов в проектировании человеко-машинных систем. Помимо психологических факторов, эргономика также в большей степени учитывает ряд других параметров, например физиологические и моторные особенности человека, показатели напряженности.

Человеко-компьютерное взаимодействие часто рассматривают как подраздел эргономики, сконцентрированный на взаимодействии человека и компьютера (подмножество взаимодействия человека с машиной). Однако, некоторые специалисты разделяют эргономику, которая занимается проектированием машин, индикаторов, то есть

в большей степени объектов физического мира, и ЧКВ, сосредоточенного на проектировании ПИ.

Когнитивный подход. В изучении и проектировании ЧКВ до, примерно, начала девяностых годов использовалась в основном парадигма когнитивной психологии. Только ее применение привело к тому, что было создано достаточно много пособий по проектированию ЧКВ. Большинство советов в них сводилось к необходимости учета ограничений, накладываемых на деятельность особенностями переработки информации человеком. Также проблемы когнитивного подхода заключались в его направленности на изучение когнитивных особенностей личности, поэтому мало освещались социальные аспекты ЧКВ.

По этим, и многим другим причинам, с середины девяностых годов к проблемам в области ЧКВ постепенно стали подходить с позиций этнографического подхода и теории деятельности.

Этнографический подход. Этнографический подход проявляет большой интерес к проектированию систем для совместной деятельности. В рамках этнографического подхода выделяется четыре основных проблемных направления: 1) проблемы включения этнографии в процесс проектирования; 2) важность этнографического подхода для описания, анализа и проектирования совместной деятельности; 3) практические отношения между этнографическими исследованиями и проектированием; 4) роль этнографии в развитии систем, поддерживающих совместную деятельность (Computer Supported Cooperative Work-систем). Однако этнографический подход не имеет точных методов и не обладает строго определенной терминологией.

Этнографический подход дает много необходимых данных для организации ЧКВ, но применять только его в качестве определяющего нецелесообразно.

Теория деятельности. В отличие от когнитивного подхода, здесь целью разработок является не просто удобство в использовании, но и полезность. Эта теория позволяет глубже анализировать

потребности пользователей и контекст их работы, с помощью изучения реальной деятельности в ее естественной среде.
Теория деятельности не опровергает достижения когнитивной психологии и не противостоит ей.

Основной задачей новой методологии было «человечное» изучение информационных систем, когда первичным является не компьютер, а человек.

5.3.2. Прикладные дисциплины и подходы

Проектирование взаимодействия (Interaction Design) — область знаний, направленная на проектирование поведения продуктов и систем, с которыми взаимодействует пользователь.

Опыт пользователя (User Experience), термин понимается в двух смыслах.

- В широком смысле User Experience объединяет множество дисциплин, связанных с проектированием и удобством использования продуктов, систем и услуг.
- В узком, определяется как «ощущение и реакцию человека, вследствие использования или предполагаемого использования продукта, системы или услуги» (стандарт ISO 9241-210).

Термин **«юзабилити»** также понимается в узком и широком смыслах.

- В широком смысле данным словом называют научно-прикладную дисциплину, служащую повышению эффективности, продуктивности и удобства использования инструментов деятельности. Это определение вызывает некоторую путаницу, при попытке установить взаимоотношения всех перечисленных дисциплин.
- В узком смысле, юзабилити — это «степень, с которой продукт может быть использован определенными пользователями при определенном контексте использования для достижения определенных целей с должной эффективностью, продуктивностью и удовлетворенностью» (стандарт ISO 9241-

11). В этой лекции слово «юзабилити» будет употребляться только в этом смысле.

User-centered design аналог человеко-ориентированного подхода, и в некотором смысле синоним User Experience. Это философия дизайна и процесс, в котором потребностям, желаниям, и ограничениям конечных пользователей продукта уделяется повышенное внимание на всех этапах создания. В рамках UCD выделяются и другие подходы, такие как Goal-centered design и Activity-centered design.

Goal-centered design прикладной подход, основополагающим моментом проектирования в котором считаются цели пользователей. Именно удовлетворение целей является критерием успешности интерфейса. При этом целью деятельности далеко не всегда является ее результат (например результатом деятельности бухгалтера является генерация годового отчета, а целью — сделать отчет без ошибок, максимально быстро и просто). Апологетом подхода в США является А. Купер.

Activity-centered design — подход основан на концепции анализа деятельности. Любая деятельность раскладывается на составляющие, учет и понимание которых необходимо для проектирования взаимодействия.

В практике подход в основном разрабатывается Д. Норманом, на основе теории деятельности сформулированной советским психологом А.Н. Леонтьевым.

Оба подходя нельзя рассматривать изолированно, их сочетание и уместное применение того или иного подхода позволяет сделать разрабатываемые интерфейсы наиболее удобными для пользователей.

5.4. Специалисты, участвующие в проектировании ПИ

Профессиональная область достаточно активно развивается, некоторые специальности выходят на первый план, некоторые

постепенно «отходят», названия одной и той же деятельности меняются. Информация в этом разделе предназначена в основном для абстрактного понимания целей и задач специалистов, нежели отражает реальную иерархическую структуру и должности специалистов в организации. Также специальности описаны в более-менее чистом виде, хотя на практике различные функции может выполнять один и тот же специалист.

Проектировщики (Interaction Designer, User Experience Designer). Проектируют интерфейс на основе имеющихся данных, отрисовывают макеты, создают прототипы, продумывают логику взаимодействия пользователя с продуктом.

Аналитики (исследователи) (Usability Analyst, Usability Tester). Проводят исследования, юзабилити-тестирования, собирают информацию предваряющую проектирование, осуществляют мониторинг рабочего продукта (напр. веб-аналитика для сайтов).

Дизайнеры графического интерфейса (Visual Designer, GUI-designer). Отрисовывают конечный вариант графического дизайна интерфейса.

Технические писатели. Обычно не входят в команду, занимающуюся проектированием интерфейса, но чрезвычайно важны в процессе, так как именно они пишут сопроводительную информацию для пользователей (руководства, справки).

Менеджеры. Управляют процессом проектирования интерфейса, ставят задачи, контролируют сроки.

Далее в лекции, все специалисты имеющие отношение к проектированию интерфейсов будут назваться «специалистами по человеко-компьютерному взаимодействию». Также таких специалистов называют «юзабилистами».

5.5. Модели интерфейса

Для понимания различий в восприятии интерфейса пользователем и разработчиком и более глубокого понимания специфики деятельности специалиста по ЧКВ, нужно знать, как все они представляют себе систему.

Модель реализации — подробности реализации программы в коде (данная модель имеет место в представлениях программиста).

Модель пользователя — представления пользователя о том как должна выглядеть и вести себя программа.

Модель представления — избранный проектировщиком способ предъявления пользователю функционирования программы.

Основная задача специалиста по ЧКВ максимально приблизить модель представления к модели пользователя. Программисту, даже если он пытается «встать» на место пользователя, сложно понять, какие потребности действительно есть у последнего, в силу, например, различного опыта использования и степени профессионализма в работе с компьютером.

5.6. Стадии проектирования ПИ

Существуют различные описания этапов создания ПИ, в зависимости от подхода в рамках которого ведется проектирование, однако все они описывают примерно один и тот же процесс. Нужно помнить, что любое описание подобного процесса в отрыве от конкретной организации, типов проектируемых интерфейсов и используемых аппаратных платформ является абстрактным. Каждая организация рано или поздно вырабатывает свою технологию проектирования и внедряет ее в процесс разработки ПО.

<u>Этапы и деятельность специалиста по ЧКВ в организации зависит от многих параметров:</u>

- разрабатывает организация интерфейсы для собственных продуктов или на заказ (во втором случае в общий процесс

включается общение с заказчиком);

- момент подключения специалиста по ЧКВ к проекту (на этапе проектирования с нуля, на этапе уже существующей версии продукта);
- тип проектируемого интерфейса (софт, веб, мобильные приложения);
- наличие аналогов (проектирование уникального продукта или повторяющего чей-то функционал);
- необходимость разработать что-то новое или укладывающееся в рамки уже существующих подходов к проектированию (в первом случае творческий подход, креативные техники, во втором руководства по стилю, рекомендации по разработке интерфейса, накопленный багаж знаний в эргономике, инженерной психологии, ЧКВ).

Этапы проектирования ПИ:

1. Сбор и анализ информации:
   - исследование пользователей (интервью, этнографическое наблюдение, анализ деятельности и задач и т.д.);
   - интервью с заинтересованными лицами (заинтересованное лицо — любой человек, обладающий полномочиями в отношении проектируемого продукта);
   - интервьюирование экспертов предметной области;
   - аудит конкурирующих и аналогичных продуктов;
   - анализ бизнес-процессов (если требуется).
2. Определение профилей пользователей (целевой аудитории), либо создание «персонажей».
3. Создание сценариев действий пользователей и требований к интерфейсу.
4. Создание инфраструктуры взаимодействия, определение следующих параметров:
   - типа приложения (веб, софт, и т.п.), способов управления;
   - функциональных и информационных элементов (определение объектов и операций);
   - функциональных групп и иерархических связей между ними.
5. Прототипирование.

6. Юзабилити-тестирование прототипа, проверка по сценариям.

Приведенные этапы описывают процесс разработки с нуля.

Нужно помнить, что так же как и процесс разработки ПО, разработка ПИ имеет итеративную структуру.

5.7. Моменты, которые необходимо учитывать при проектировании ПИ

Уровень компьютерной грамотности пользователей. Пользователей, пусть и весьма условно, можно поделить на три неравные группы:

- «новички» — люди, совсем недавно освоившие компьютер, проводящие за ним немного времени. Сегмент относительно невелик, так как новички достаточно быстро переходят в следующую группу;
- «середняки» имеют достаточный опыт взаимодействия с компьютером, который позволяет им нормально работать и выполнять собственные задачи. Большая часть пользователей остаются в этом сегменте, не переходя в следующий;
- «профессионалы» прекрасно разбираются в технологиях, обычно имеют отношения к сфере IT. Разработчики ПО представляют именно эту группу. При изменениях в продуктах, пользователи из этой группы могут временно переходить в разряд «середняков», а после освоения нововведений возвращаться в группу «профессионалов».

С уровнем технической грамотности связано несколько проблем:

- первичная адаптация (обучение «новичков»);
- переход «новичков» в разряд «середняков» (удаление тех средств, которые использовались в начале для адаптации и обучения);
- переход из «середняков» в «профессионалы» (свертывание привычных действий, например использование горячих клавиш).

Интернационализация. При проектировании интерфейса, который будет использоваться в разных странах, необходимо внимательно относится к учету национальных особенностей, не говоря уже о поддержке национального языка.

Например, меры в разных странах представлены в различных единицах, которые должны быть адаптированы под конкретную страну.

В процессе адаптации текстов необходим специалист-носитель языка, который сможет не только оценить грамотность перевода, но и обнаружить неадекватные для данной национальности слова и фразы. Наглядным примером, правда из области кинематографа, может послужить адаптация названий одних и тех же фильмов или товаров для разных национальностей.

Итак, в процессе разработки интерфейсов для использования в других странах необходимо учитывать следующие моменты.

Для текстовых элементов:

- международные аббревиатуры;
- акронимы;
- описательный текст;
- выделение мнемоник (средств, помогающих облегчить запоминание);
- грамматика персонализации;
- двунаправленность языков (написание слева-направо, справа-налево);
- длинна текстов;
- правила использования заглавных букв;
- вид заголовков колонок;
- юмор (табуированные темы).

Для изображений:

- вид иконок (некоторые привычные для нас жесты, изображаемые на картинках, могут являться оскорбительными для людей других национальностей)
- использование символов;
- использование цветов.

Для действий:

- использование клавиш быстрого доступа;
- назначения комбинации клавиш;
- принципы сортировки информации.

Для форматов:

- денежных единиц;
- телефонных номеров;
- размеров;
- адресов;
- дат;
- чисел;
- времени.

Аксессабилити. В США, для того, чтобы программный продукт мог быть закуплен государственной организацией, он должен удовлетворять требованиям accessibility (доступности). Учет требований accessibility позволяет использовать продукт людям с ограниченными возможностями. Крупные компании, покупающие софт имеют свои чек-листы, по которым проверяется соответствие интерфейса требованиям. Пример такого чек-листа можно посмотреть по адресу http://www.justice.gov/crt/508/archive/oldsoftware.html.

На практике особые потребности удовлетворяют в тех программных продуктах, в целевую аудиторию которых входят пользователи с различными нарушениями. Либо в тех, которые разработаны специально для данных категорий пользователей.

Патологии зрения: До 8% мужского населения страдает дальтонизмом (нарушением цветового зрения). Среди них выделяются люди, страдающие полной и частичной цветовой слепотой. Первые воспринимают окружающее только в черно-белом цвете. При частичной цветовой слепоте могут не восприниматься красный, зеленый или синий цвета. Эти параметры необходимо учитывать при проектировании цветовой схемы

интерфейса, для того чтобы он оставался легко «читаемым» для людей с и нормальным и патологичным типами цветового зрения.

Патологии слуха: Их необходимо учитывать, если в интерфейсе существуют аудио-оповещения не дублирующиеся другими способами.

Патологии двигательного аппарата (нарушения моторики): Необходимо учитывать при проектировании способов ввода. В случае возникающих у пользователей трудностей, использовать альтернативные способы ввода.

Учет возрастных особенностей пользователей. При проектировании различных возрастных категорий необходимо учитывать показатели:

- зрения (чем старше пользователь, тем зрение хуже, ниже цветочувствительность и цвето-дифференциация);
- слуха (постепенное снижение к старости);
- моторных возможностей (ухудшение моторики с возрастом).

Мультимедиа (видео, аудио, слайдшоу). Мультимедиа-элементы должны обладать следующими ключевыми характеристиками:

- представлять и взаимодействовать со множеством объектов одновременно;
- одновременно представлять множество видов объектов;
- обеспечивать преимущественно прямое манипулирование данными и объектами;
- скрывать всю сложность средств информации и технологии от пользователей.

5.8. Два слова об «интуитивном интерфейсе»

Достаточно часто можно услышать загадочные слова «интуитивный интерфейс». Под этим понятием подразумевается, что можно спроектировать что-либо настолько хорошо, что

человек, в первый раз увидев «это» (не важно софт или предмет), будет знать как его использовать.

Увы, подобный идеал недостижим. Человек из изолированного племени в джунглях Амазонии, увидев простую вешалку для одежды, с трудом догадается, для чего она предназначена. В его прошлом опыте не было вешалок для одежды.

Когда кто-либо говорит об «интуитивном интерфейсе», это нужно понимать как «интерфейс, соответствующий предыдущему опыту пользователя». Только в том случае, если в жизни пользователя уже есть опыт взаимодействия с аналогичным или похожим продуктом, либо с классом подобных продуктов, будет присутствовать «интуитивное понимание».

5.9. Специфика проектирования для мобильных устройств

При проектировании для мобильных устройств (в частном случае планшетников и смартфонов) необходимо учитывать, что пользователь общается с устройством в основном с помощью прямого манипулирования объектами интерфейса.

С одной стороны это воплощение метафоры взаимодействия с предметами реального мира, что являлось идеалом проектирования ПИ многие годы. С другой, ряд трудностей в тех ситуациях, когда невозможно подобрать однозначную аналогию или простое действие для совершения необходимой операции.

5.10. Выводы

В лекции были рассмотрены основы человеко-ориентированного подхода к проектированию ПИ. Описаны дисциплины, в рамках которых осуществляются разработки методов и методик проектирования. Рассмотрены роли специалистов в процессе создания ПИ, и его этапы. Обозначены основные задачи специалистов в сфере ЧКВ. Описаны основы проектирования интерфейсов для мобильных устройств.

5.11. Контрольные вопросы

1) Для какого подхода в проектировании человеко-машинных систем характерно отношение к человеку как звену технической системы?
   1. системно-технический
   2. антропоцентрический
   3. машиноцентрический
   4. человеко-ориентированный

2) В каком подходе к созданию ЧКВ-систем делается упор на необходимость учета ограничений, накладываемых на деятельность человека его особенностями переработки им информации?
   1. этнографическом
   2. когнитивном
   3. анализе деятельности

3) Что такое модель представления?
   1. Подробности реализации программы в коде.
   2. Избранный проектировщиком способ предъявления пользователю функционирования программы.
   3. Представления пользователя о том как должна выглядеть и вести себя программа

4) Какой этап при разработке ПИ с нуля является первым?
   1. создание персонажей
   2. юзабилити-тестирование
   3. сбор и анализ информации, исследования
   4. создание сценариев

5) На каком уровне компьютерной грамотности находится большинство пользователей?
   1. новички
   2. на среднем
   3. профессионалы

6) Как правильно трактовать смысл понятия «интуитивный интерфейс»?
   1. Интуиция пользователя.
   2. Опыт пользователя.
   3. Интеллект пользователя.

Список литературы

1. Купер А., Психбольница в руках пациентов, Спб.: Символ-Плюс, 2004.
2. Купер А., Об интерфейсе. Основы проектирования взаимодействия, Спб.: Символ-Плюс, 2009.
3. Логунова О.С., Ячиков И.М., Ильина Е.А., Человеко-машинное взаимодействие: теория и практика, Ростов-на-дону, Феникс, 2006.
4. Магазанник В.Д., Человеко-компьютерное взаимодействие, Логос, 2011.
5. Мандел Т., Дизайн интерфейсов, М.: ДМК, 2005.
6. Норман Д.А., Дизайн привычных вещей, Вильямс, 2006.
7. Сергеев С.Ф., Инженерная психология и эргономика, М.: НИИ школьных технологий, 2008.

6. Лабораторная работа № 1 «Использование веб-камеры и пальцевого интерфейса сенсорного экрана»

6.1. Цель лабораторной работы

Демонстрация процесса разработки практического приложения, позволяющего рисовать пальцем на снимке с вебкамеры.

6.2. Инструкция по выполнению лабораторной работы

Основное устройство ввода на планшетном компьютере – сенсорный экран или кратко тачскрин. Первоначально тачскрин был вынужденной заменой широко распространенной «мыши», пользовательские интерфейсы для него требовали высокой точности клика, для чего требовался стилус. Затем произошло усовершенствование интерфейсов, их огрубление под пальцевый ввод, необходимость в стилусе отпала, но частичная взаимозаменяемость тачскрина и «мыши» сохраняется.

В этой лабораторной работы мы будем использовать тачскрин как холст художника – наносить пальцами штрихи на изображение. Для этого необходимы базовые знания языка C++, знакомство с основами фреймворка *Qt* и библиотекой MeeGoTouch. В качестве одной из составляющих приложения будут использованы материалы лабораторной работы «Захват видеоизображения» из первой части курса. Также необходим планшет, работающий под управлением MeeGo. Мы использовали MeeGo 1.1.99 Tablet на планшете 3Q TU1102T.

На эмуляторе или виртуальной машине поддержки тачскрина скорее всего не будет, функции взаимодействия с пользователем возьмет на себя «мышь» операционной системы.

Задача состоит из четырех частей – получение кадра от вебкамеры, отображение полученного кадра в пользовательском интерфейсе, использование информации от тачскрина для рисования и сохранение полученного рисунка в файл. Работа с вебкамерой подробно освещена в первой части курса, мы вернемся к ней в конце.

Рассмотрим задачу отображения графической информации в GUI. Библиотека MeeGoTouch содержит специальный виджет – MImageWidget – для этой цели. Достаточно добавить этот виджет в приложение и назначить ему изображение методом setImage(). Данный метод может принимать в качестве параметра QImage или QPixmap. Оба эти класса *Qt* предназначены для хранения изображений, но первый ориенитрован на чтение/запись изображений в различных форматах и попиксельный доступ к изображению для редактирования, а второй оптимизирован для вывода изображения на экран. Получить от MImageWidget можно только QPixmap, но не QImage. Естественно, существует возможность конвертации изображения из одного хранилища в другое. Кроме того, в *Qt* существует класс QBitmap, позволяющий хранить особый вид изображений – битовые маски, а также класс QPicture, который хранит изображение в форме последовательности команд для отрисовки.

Следующим шагом настройки MImageWidget будет установка его размеров в соответствии с размерами картинки.

```
widget.setMaximumSize(image.size());
widget.setMinimumSize(image.size());
widget.setPreferredSize(image.size());
```

Если этого не сделать, то компоновщик в некоторых ситуациях (например, изменение ориентации устройства) может изменить размеры виджета и координаты внутри виджета перестанут совпадать с координатами изображения. Возможно преобразовывать координаты на лету, но мы вместо этого просто фиксируем размер виджета. В приложении создан класс MyWidget – наследник MImageWidget, к которому относится все вышенаписанное.

Теперь надо получить информацию от тачскрина. Большая часть информации внутри *Qt* передается в виде событий – наследников класса QEvent, попадающих в соответствующую очередь, а через нее – к виджетам и другим элементам приложения. События тачскрина относятся к классу QTouchEvent и имеют всего 3 типа – QEvent::TouchBegin (начало прикосновения), QEvent::TouchUpdate (движение по тачскрину), QEvent::TouchEnd (окончание прикосновения. По умолчанию они не доходят до нашего виджета в неизменном виде, система преобразует их к событиям более высокого уровня (например, QGestureEvent) и обрабатывает в другом месте. Для получения события тачскрина надо вызвать метод setAcceptTouchEvents(true).

Чтобы обрабатывать события, необходимо переопределить обработчик bool MyWidget::event(QEvent *event). Такой обработчик есть у каждого виджета и вообще может быть реализован у любого наследника QObject. В новом обработчике мы проверяем тип события, и если оно относится к тачскрину – выполняем приведение типов, чтобы получить доступ к специфичной информации:

```
QTouchEvent *touchevent =
static_cast<QTouchEvent*>(event);
```

Для приведения типов мы используем конструкцию static_cast, использование dynamic_cast не рекомендуется, как

и приведения в стиле чистого C. После обработки события следует сообщить очереди событий, что оно уже обработано и в дальнейшей обработке не нуждается:

```
touchevent->accept();
return true;
```

Для тех событий, которые не относятся к тачскрину, вызывается обработчик класса-родителя:

```
return MImageWidget::event(event);
```

Каждое событие тачскрина содержит список точек touchPoints(), обычно состоящий из одной точки, которые мы будем использовать для рисования на изображении – просто соединяя предыдущую и текущую точку линией. В приложении это происходит прямо в обработчике событий – виджет выдает текущий QPixmap, на который с помощью QPainter наносится линия и обновленный QPixmap устанавливается виджету. Конечно, это не самое оптимальное с точки зрения производительности и организации программы решение. Правильнее было бы вынести из обработчика все ресурсоемкие операции, такие как обновление изображения, или вообще перенести их в специальный обработчик paintEvent(), отвечающий за отрисовку графических виджетов. Чтобы сэкономить ресурсы, отрисовка производится не для каждой следующей точки, а только в случае, если расстояние между точками превышает определенный порог (или в случае, когда эта точка последняя).

К сожалению, несмотря на перехват событий тачскрина в MyWidget, они все равно продолжают попадать в компоновщик в виде gestures, и одновременно с рисованием на виджете происходит его движение внутри главного окна приложения. Чтобы избавится от этого нежелательного эффекта, необходимо также перехватить события gesture, для этого надо использовать метод

```
grabGesture(Qt::PanGesture);
```

В уже созданном обработчике bool MyWidget::event(QEvent *event) надо добавить перехват события QEvent::Gesture, с ним ничего не надо делать по существу, просто отметить как обработанное, чтобы оно не попало к компоновщику.

В конце работы изображение может быть сохранено в файл, для этого в *Qt* есть специальный класс QImageWriter. Он автоматически определяет формат сохраняемого файла по заданному расширению.

Перед сохранением QPixmap, полученный от виджета, необходимо сконвертировать в QImage. Все это реализовано в слоте saveImage().

Вернемся к захвату видеокадра, который будет служить фоном для рисунка. Захват реализован в виде отдельного приложения v4l2grab, работающего напрямую с видеоподсистемой ядра MeeGo, которое можно собрать непосредственно на планшете, без использования *Qt* Creator. Установите зависимости

```
zypper install libjpeg-devel
```

скомпилируйте исполняемый файл

```
gcc v4l2grab.c -o v4l2grab -ljpeg
```

и перенесите его в /usr/bin

```
mv v4l2grab /usr/bin
```

Теперь v4l2grab можно использовать в главном приложении с помощью класса QProcess, который позволяет запускать внешние приложения из *Qt*. Целесообразно установить таймаут выполнения внешнего приложения, в данном случае это 3000 миллисекунд, т.е. 3 секунды. Созданное v4l2grab изображение сохраняется в файл /tmp/webcam_image.jpg, откуда потом его читает QImageReader:

```
v4l2grab.start("v4l2grab -W 800 -H 600 -d
/dev/video0"
" -o /tmp/webcam_image.jpg");
v4l2grab.waitForFinished(3000);
```

```
QImageReader jpg("/tmp/webcam_image.jpg");
jpg.read(&image);
```

Фоновое изображение также можно создать программно, для этого используется уже упомянутый QPainter, обладающий богатым набором графических примитивов.

Для удобства пользователя приложение, помимо виджета с изображением, оснащено тремя кнопками с помощью компоновщика и механизма слотов и сигналов.

6.3. Задания для самостоятельной работы

1. Добавить возможность изменять параметры пера (цвет, толщину, ...) при рисовании
2. Добавить диалоги для открытия файла изображения с диска и записи на диск
3. Использовать вместо утилиты v4l2grab компоненту Multimedia библиотеки *Qt* Mobility.

6.4. Выводы

6.5. Контрольные вопросы

1) Какой класс *Qt* не предназначен для хранения изображений:
   1. QPainter
   2. QImage
   3. QPixmap

2) Для каких задач оптимизирован класс QPixmap:
   1. Ввод/вывод в файл
   2. Доступ к пикселям
   3. Отображение на экране

3) Какой виджет библиотеки MeeGoTouch предназначен для отображеия на экране рисунков:
   1. QImageWidget
   2. QPixmapWidget
   3. MImageWidget
   4. MWidget

4) Какой способ приведения типов рекомендовано использовать в *Qt*:
   1. dynamic_cast <>
   2. static_cast <>
   3. (type *)

5) Что представляет собой метод bool event(QEvent *event):
   1. Обработчик событий
   2. Источник событий
   3. Фильтр событий

6) Для чего предназначен метод grabGesture():
   1. Метод очищает очередь gesture
   2. Метод включает обработку gesture в виджете
   3. Метод возвращает последний введенный gesture

7) Какой класс *Qt* предназначен для сохранения рисунка в файл:
   1. QPixmapFile
   2. QImageWriter
   3. QPictureStream
   4. QFile

8) Как можно использовать класс QProcess:
   1. Как абстрактный класс для создания обработчиков данных

2. Для доступа к внутренним настройкам *Qt*
3. Для запуска внешних программ

9) Как корректно завершить отрисовку классом QPainter:
   1. Никак, отрисовка продолжается все время работы приложения
   2. Методом end()
   3. Никак, управление классом QPainter явно недоступно программисту

Список литературы

1. MeeGo Tablet Developer Preview.
2. https://meego.com/downloads/releases/1.2/meego-tablet-developer-preview
3. MeeGo Touch. http://apidocs.meego.com/git-tip/mtf/
4. Gestures Programming. http://doc.qt.nokia.com/4.7-snapshot/gestures-overview.html
5. Qt Mobility 1.2 Multimedia API. http://doc.qt.nokia.com/qt-mobility-snapshot/multimedia.html

7. Лабораторная работа № 2 «Использование Qt Mobility для хранения контактов и рассылки SMS»

7.1. Цель лабораторной работы

Демонстрация процесса разработки практического приложения для хранения контактов и автоматической рассылки SMS приглашений на день рождения по контактам адресной книги с фильтром по диапазону возрастов контактов.

7.2. Инструкция по выполнению лабораторной работы

Для работы с сообщениями и контактами в MeeGo есть готовый интерфейс – *Qt* Mobility, компоненты Contacts и Messaging. Для использования этого интерфейса необходимо добавить в файл проекта строки:

```
CONFIG += mobility
MOBILITY = contacts messaging
```

В исходный текст программы надо добавить макрос QTM_USE_NAMESPACE, который выберет соответствующее пространство имен. После этого разработка не отличается от разработки обычного *Qt*-приложения. *Qt* Mobility пока еще является довольно изменчивым интерфейсом, описываемый далее пример был создан для MeeGo Netbook 1.0 с *Qt* Mobility 1.0.0, при работе с более поздними версиями (уже есть 1.2), возможно, потребуются небольшие изменения в коде. Используемые компоненты *Qt* Mobility содержат довольно

много классов, наиболее важными для понимания являются: QContact, QContactManager, QContactDetailRangeFilter, QMessage, QMessageService, QMessageManager.

Спроектируем нашу программу следующим образом – у нее будет поле ввода сообщения для рассылки, два поля ввода возраста (минимальный и максимальный) для выборки, окно для отображения выбранного списка контактов и две кнопки – сделать выборку и разослать сообщение. Функциональность будет реализована в классе SenderWidget, наследнике QWidget. Опишем в нем два слота – select() и send() – они будут вызываться при нажатии соответствующих кнопок. Виджеты для ввода данных сделаем полями класса, чтобы иметь к ним доступ из слотов. Также добавим поле класса manager, в котором будем хранить указатель на используемый менеджер контактов.

Для работы с контактами опишем небольшой API.

```
QList<QContact> selectContacts(int from, int till);
```

Функция получает от *manager* список контактов в заданном интервале возрастов – например, от 18 до 23 лет.

```
void sendContacts(QString message, QList<QContact>
contacts);
```

Функция отправляет сообщение *message* каждому контакту в списке.

```
void showContacts(QList<QContact> contacts);
```

Функция отображает список контактов в окне вывода.

```
QContact makeContact(QString firstName, QString
lastName, QString phone, QDate birth);
```

Вспомогательная функция, которая создает новый контакт на основе переданных ей параметров.

Как нетрудно видеть, используя это API, задача решается тривиально, осталось только его реализовать. Начать можно с выбора менеджера, в котором будут храниться контакты. Список идентификаторов всех менеджеров можно получить вызвав метод QContactManager::availableManagers(). Из

полученного списка строк нужно удалить все неисправные менеджеры (они называются "invalid"). Для работы мы возьмем самый первый доступный менеджер, создав его по идентификатору, в нашем случае это "memory". Для других платформ идентификатор может оказаться другим, но так как менеджер создается только один раз в начале программы и из списка доступных менеджеров, то на работоспособность примера это не повлияет. Затем добавляем в менеджер тестовые контакты

```
manager->saveContact(&makeContact( "Alice", "Smith",
"+79111234567", QDate(1990, 1, 1)));
```

Функция makeContact() появилась из-за того, что создание нового контакта в *Qt* Mobility достаточно громоздко. Например, для добавления телефонного номера в контакт надо выполнить три шага:

```
QContactPhoneNumber phonenumber;
phonenumber.setNumber(phone);
contact.saveDetail(&phonenumber);
```

Для осуществления выборки в *Qt* Mobility следует использовать фильтры на основе QContactFilter. Конечно, никто не может запретить программисту извлечь все контакты и отфильтровать их самостоятельно, но эта процедура может оказаться очень неэффективной. Для нашей цели наиболее подходит фильтр QContactDetailRangeFilter, который выбирает значения в заданном интервале. Перед использованием надо указать поле, в нашем случае дату рождения:

```
filter.setDetailDefinitionName(
QContactBirthday::DefinitionName,
QContactBirthday::FieldBirthday);
а также интервал возрастов:
filter.setRange(QDate::currentDate().addYears(-till),
QDate::currentDate().addYears(-from));
```

Обратите внимание, что в контакте хранится не возраст, который меняется со временем, а дата рождения, которая постоянна, поэтому для настройки фильтра мы вычисляем

интервал дат рождения для желаемых нами возрастов, для чего из текущей даты вычитаем возраст.

Передав настроенный фильтр менеджеру, получаем искомое:

```
QList<QContact> result = manager->contacts(filter,
QList<QContactSortOrder>(), QStringList());
```

Последние два параметра пусты и не несут смысловой нагрузки.

Для отображения контактов достаточно пробежаться по их списку и получить атрибуты (details) каждого контакта. Например, для телефонного номера это выглядит так

```
contact.detail<QContactPhoneNumber>().number()
```

Для отправки сообщения к предыдущей процедуре надо добавить использование Messaging. Сначала нужно выбрать подходящий для отправки SMS аккаунт, для этого воспользуемся QMessageManager:

```
QMessageManager messageManager;
QMessageAccountId id;
foreach (id, messageManager.queryAccounts()) {
    QMessageAccount account(id);
    if (account.messageTypes() & QMessage::Sms)
        break;
}
```

Теперь id содержит идентификатор аккаунта для отправки и можно использовать QMessageService:

```
QMessageService service;
QMessage sms;
sms.setParentAccountId(id);
sms.setType(QMessage::Sms);
sms.setBody("My message");
QMessageAddress to(QMessageAddress::Phone,
"1234567");
sms.setTo(to);
service.send(sms);
```

Функция send() не отправляет SMS сама, а ставит ее в очередь на отправку. Кроме того, важно, чтобы в системе был

доступен аккаунт для отправки. Например, на нетбуке с MeeGo Netbook нет штатных средств для отправки SMS и нет такого аккаунта.

Однако мы добьемся работоспособности приложения на нетбуке, используя внешний модем и oFono через DBus. Во-первых, необходимо подключить модем к oFono. Для модема ZTE MF100, который использовали мы, достаточно добавить к конфигурационный файл /lib/udev/rules.d/97-ofono.rules следующие две строки:

```
ATTRS{idVendor}=="19d2",
ATTRS{idProduct}=="0016",
          ENV{OFONO_IFACE_NUM}=="01",
          ENV{OFONO_ZTE_TYPE}="modem"
ATTRS{idVendor}=="19d2",
ATTRS{idProduct}=="0016",
          ENV{OFONO_IFACE_NUM}=="02",
          ENV{OFONO_ZTE_TYPE}="aux"
```

Для других модемов исправление конфиг-файлов может и не потребоваться. После настройки модем появляется в списке доступных устройств oFono.

oFono предоставляет свои сервисы по интерфейсу DBus. В Qt существует компонента, предоставляющая возможность работы с DBus, но мы используем более простой вариант – скрипт на Python, который прилагается к oFono и умеет посылать SMS. Вызовем скрипт как внешнее приложение:

```
QProcess ofono;
ofono.start(QString("/usr/lib/ofono/test/send-
sms %1 \"%2\"")
.arg(number).arg(message));
ofono.waitForFinished(1000);
```

Сборку приложения можно осуществить прямо в MeeGo. Установим зависимости

```
yum install make gcc-c++ qt-devel qt-mobility-
devel
```

Собираем с помощью qmake; make. Запускаем приложение, будут созданы два тестовых контакта с возрастом 21 и 31 год. Введите интервал возраста и нажмите кнопку «Select», в списке адресатов должны отобразиться контакты. Введите сообщение и нажмите кнопку «Send», приложение попытается разослать введенное сообщение всем контактам из списка.

7.3. Задания для самостоятельной работы

1. Сделайте недоступной кнопку «Send», если с момента последнего нажатия кнопки «Select» в полях ввода возраста произошли изменения.
2. Добавьте к выборке контакта по возрасту выборку по имени
3. Замените телефон на адрес электронной почты и отсылайте сообщение по почте, а не по SMS
4. При компиляции строки

```
manager->saveContact( &makeContact(...));
```

компилятор gcc *4.4.2* выдает предупреждение «*warning: taking address of temporary*». Объясните, почему это происходит и есть ли реальная проблема. Измените код, чтобы предупреждение исчезло.

7.4. Выводы

В этой лабораторной работе рассмотрен пример рассылки SMS-сообщений по списку контактов.

7.5. Контрольные вопросы

1) Какой компонент *Qt* Mobility не используется в приложении:
   1. Contacts
   2. Messaging
   3. Publish and Subscribe

2) Какое пространство имен использует *Qt* Mobility:

1. Свое собственное
2. Пространство имен совпадает с *Qt*
3. Пространство имен не используется

3) Для чего используется метод QLineEdit::setInputMask():
   1. Задает шаблон допустимых значений
   2. Позволяет заблокировать ввод нового значения
   3. Устанавливает цвет фона окна ввода

4) Допускает ли *Qt* вложенные layouts:
   1. Нет
   2. Допускает только если их типы совпадают
   3. Да

5) Для чего предназначен класс QContactManager:
   1. Управляет отображением контактов на мобильном устройстве
   2. Ускоряет работу с удаленными контактами путем кэширования
   3. Обеспечивает доступ к хранилищу контактов.

6) Какой объект служит идентификатором менеджера контактов QContactManager:
   1. QContactManagerId
   2. QString
   3. unsigned int

7) Что делает метод QContact::saveDetail():
   1. Сохраняет подробное текстовое описание контакта
   2. Сохраняет произвольный атрибут контакта
   3. Сохраняет контакт в XML для последующего экспорта

8) Какой фильтр следует использовать для поиска в заданном интервале дат:
   1. QContactDetailRangeFilter
   2. QContactDetailFilter
   3. QContactDateFilter
   4. QContactRelationshipFilter

9) Зачем приложение запрашивает список аккаунтов у QMessageManager:
   1. Для поиска среди ассоциированных с аккаунтом контактов
   2. Ищет аккаунт с возможностью отправки SMS
   3. Чтобы активировать все аккаунты

10) Что делает метод QMessageService::send():
   1. Ставит сообщение в очередь на отправку
   2. Отправляет сообщение
   3. Отправляет сообщение и ожидает подтверждение получения

11) По какому интерфейсу доступен oFono:
   1. EABI
   2. DBus
   3. SOAP

Список литературы

1. *Qt* Mobility/Contacts. http://apidocs.meego.com/1.0/qtmobility/contacts.html
2. *Qt* Mobility/Messaging. http://apidocs.meego.com/1.0/qtmobility/messaging.html

8. Лабораторная работа № 3 «Использование датчика ориентации для управления пользовательским интерфейсом»

8.1. Цель лабораторной работы

Демонстрация процесса разработки практического приложения.

8.2. Инструкция по выполнению лабораторной работы

Отличительная особенность современного смартфона или планшетного компьютера – наличие большого количества разнородных датчиков, предназначенных для определения положения и скорости устройства, освещенности и других параметров. В отличие от традиционных датчиков, например, температуры процессора или скорости вентилятора, они предназначены в первую очередь для уровня приложений, а не операционной системы. В отличие от внешних датчиков, например, GPS, они являются неотъемлемой частью устройства и не могут использоваться самостоятельно.

Наша задача – научиться работать с одним самых распространенных датчиков, датчиком ориентации, который позволяет определить, «на каком боку» в данный момент «лежит» устройство. Для этого необходимы базовые знания языка C++, знакомство с основами фреймворка *Qt* и двумя библиотеками *Qt* Mobility и MeeGoTouch. Самое важное требование – необходимо устройство (смартфон или планшет) с датчиком ориентации, работающее под управлением MeeGo. Мы использовали MeeGo 1.1.99 Tablet на планшете 3Q TU1102T. Поскольку на эмуляторе или

виртуальной машине информация от датчиков будет недоступна, разработку придется вести на реальном оборудовании.

Сначала рассмотрим интерфейс MeeGo для работы с датчиками, который реализован как компонента sensors библиотеки *Qt* Mobility. Как обычно, для использования этой компоненты в файл проекта необходимо добавить строки

```
CONFIG += mobility
MOBILITY = sensors
```

В исходных текстах также надо употреблять макрос QTM_USE_NAMESPACE.

Основным классом для работы с датчиками является QSensor. От него унаследована масса классов для работы с различными типами датчиков, например, QOrientationSensor или QAccelerometer. Чтобы получить все доступные системе типы датчиков, используется статический метод QSensor::sensorTypes(), который возвращает список строк с названиями типов, строки упакованы в классы QByteArray. В свою очередь, для каждого типа с помощью статического метода QSensor::sensorsForType() можно получить список идентификаторов подходящих датчиков, также в виде списка строк в формате QByteArray.

Зная тип и идентификатор датчика, можно создать экземпляр соответствующего класса:

```
QSensor sensor(type);
sensor.setIdentifier(identifier);
```

Или явно указав тип датчика, если он известен заранее:

```
QOrientationSensor sensor;
sensor.setIdentifier(identifier);
```

Заметим, что созданный объект не получает никакого эксклюзивного контроля над датчиком, он просто дает возможность приложению получать от датчика данные и, возможно, настраивать некоторые его параметры. Драйвер датчика действует в ядре MeeGo, специальная библиотека координирует доступ к датчикам со стороны программ. Можно создать несколько QSensor для получения данных от одного датчика.

Метод QSensor::connectToBackend() подключает объект к датчику и позволяет проверить, что датчик с таким идентификатором действительно существует и работоспособен.

Для доступа к данным датчика предназначен класс QSensorReading и его производные классы для разных типов датчиков. Они предоставляют удобный интерфейс для специфических данных.

Мы будем работать с датчиком ориентации, которому соответствует класс QOrientationSensor и класс данных QOrientationReading. Возможных показателей этого датчика всего 7, вот они:

```
QOrientationReading::Undefined
QOrientationReading::TopUp
QOrientationReading::TopDown
QOrientationReading::LeftUp
QOrientationReading::RightUp
QOrientationReading::FaceUp
QorientationReading::FaceDown
```

Последние два значения служат для определения того, что устройство лежит на какой-то поверхности вверх или вниз экраном. Наш планшет данные значения не предоставляет и таким образом мы оперировали четырьмя содержательными значениями – TopUp, TopDown, LeftUp, RightUp.

Важным является способ получения данных от датчика. Наиболее простым и самоочевидным способом является поллинг – периодический опрос датчика. Новые данные при этом обрабатываются в соответствии с логикой программы, старые данные или их отсутствие игнорируется. Главный недостаток поллинга – его вычислительная неэффективность, постоянное расходование ресурсов на бесполезные операции. Кроме того, современные приложения обычно строятся как событийно-управляемые (нет непрерывного потока управления, есть множество обработчиков различных событий) , и для организации классического поллинга нужно идти на технические ухищрения.

Более эффективен опрос датчика по прерываниям или иным способам нотификации. В этом случае датчик при готовности новых данных генерирует прерывание, и его обработчик забирает данные. Механизм слотов и сигналов *Qt* идеально подходит для этой ситуации.

Для того, чтобы получить оповещение о готовности данных датчика, достаточно связать сигнал датчика readingChanged() с каким-либо обработчиком и активировать получение данных методом start(). Важно понимать, что и создание объекта QSensor, и начало приема данных совсем необязательно как-то повлияют на физический датчик или его ядерный драйвер. Например, датчик ориентации в MeeGo постоянно включен и поставляет данные приложениям на основе MeeGoTouch. Когда мы запускаем наше тестовое приложение и выполняем вышеописанные действия, мы просто добавляем еще одного слушателя к уже работающему датчику.

Есть датчики, которые работают с высокой частотой, и даже механизм слотов и сигналов оказывается для них слишком тяжелым. Для решения этой проблемы применяются фильтры, создаваемые программистом, которые решают, достаточно ли полученные данные важны, чтобы оповещать о них слушателей. В нашем примере фильтры не используются.

В слоте readSensor(), который был связан с сигналом датчика readingChanged(), мы получаем свежее значение датчика ориентации. Для наглядности оно будет использоваться для вращения одного из двух больших смайлов на переднем плане приложения – красный смайл будет ориентироваться с помощью стандартных средств, синий смайл будет ориентироваться вручную по данным датчика.

Поскольку приложение написано с использованием библиотеки MeeGoTouch, его GUI имеет встроенную поддержку датчика ориентации, т. е. поворачивается вслед за поворотами планшета. Пока не нажата кнопка «Find sensors», оба смайла поворачиваются одинаково. После нажатия кнопки и активации датчика ориентации красный смайл продолжает вращаться, а синий становится

неподвижным относительно корпуса планшета. Для вращения изображения применяются стандартный способ *Qt* – класс QPainter.

Для дальнейшей работы с датчиками целесообразно ознакомится с примером *Qt* Mobility «Sensor Explorer», входящим в пакет qt-mobility-examples. В MeeGo его расположение /usr/lib/qtmobility/examples/sensor_explorer. Sensor Explorer позволяет удобно просмотреть имеющиеся в системе датчики, их параметры и данные.

8.3. Задания для самостоятельной работы

1. Создать приложение для работы с двумя датчиками одновременно.
2. Создать *Qt*-приложение без использования MeeGoTouch, так чтобы его пользовательский интерфейс реагировал на датчик ориентации.

8.4. Выводы

В этой лабораторной работе рассмотрен пример рассылки SMS-сообщений по списку контактов.

8.5. Контрольные вопросы

1) В какой библиотеке реализовано API датчиков:
   1. *Qt* Embedded
   2. *Qt* Mobility
   3. MeeGoTouch
   4. libsensor

2) Класс QAccelerometer – это:
   1. Наследник класса QSensor
   2. Непосредственный наследник класса QObject
   3. Базовый класс для класса QaccelerometerReading

3) Что позволяет получить класс QSensor:
   1. Список всех датчиков независимо от типа
   2. Указатель на последний сработавший датчик
   3. Список всех типов датчиков
   4. Указатель на системную таблицу датчиков

4) Каковы данные от датчика ориентации QOrientationReading:
   1. Виртуальные
   2. Непрерывные
   3. Дискретные
   4. Волатильные

5) Для чего предназначен QSensor::connectToBackend():
   1. Для связи сигнала со слотом backend()
   2. Для подключения объекта к датчику и проверки его работоспособности
   3. Это обертка системного вызова.

6) Что является главным преимуществом поллинга:
   1. Вычислительная эффективность
   2. Ориентация на событийно-управляемое программирование
   3. Простота реализации

7) Какой сигнал сообщает о появлении новых данных от датчика:
   1. activeChanged()
   2. readingChanged()
   3. availableSensorsChanged()

8) Какие приложения имеют встроенную поддержку датчика ориентации на уровне GUI:

1. Таких приложений нет
2. Любые приложения MeeGo
3. Любые приложения *Qt*
4. Любые приложения MeeGoTouch

9) Что такое Sensor Explorer:
   1. Пример библиотеки *Qt* Mobility
   2. Штатное приложение MeeGo для поиска датчиков
   3. SDK для создания новых датчиков

Список литературы

1. MeeGo Tablet Developer Preview. https://meego.com/downloads/releases/1.2/meego-tablet-developer-preview
2. MeeGo Touch. http://apidocs.meego.com/git-tip/mtf/
3. Qt Mobility 1.2 Sensors API. http://doc.qt.nokia.com/qt-mobility-snapshot/sensors-api.html
4. Sensor Explorer. http://doc.qt.nokia.com/qt-mobility-snapshot/sensors-sensor-explorer.html

9. Лабораторная работа № 4 «Обеспечение положительного User Expirience/Usability в сложных пользовательских интерфейсах в MeeGoTouch»

9.1. Цель лабораторной работы

Демонстрация процесса разработки практического приложения.

9.2. Введение

User Experience (опыт пользователя, UX) – неотъемлемая составляющая разработки программного продукта для массовой аудитории. Не обеспечив положительный опыт пользователя, нельзя добиться успеха на рынке.

Согласно стандарту ISO 9241-210 User Experience – «ощущения и реакция человека, вследствие использования или предполагаемого использования продукта, системы или услуги». UX характеризует не саму программу или ее пользовательский интерфейс, а восприятие его пользователем, т.е. необходимо также учитывать предыдущий опыт пользователя и контекст использования. Причем имеет значение не только факт использования программы, но и отказ от использования по каким-либо причинам тоже.

Обеспечение положительного опыта использования – задача не столько программиста, сколько специалистов по маркетингу, usability и т. п., они выступают постановщиками задачи для программиста. Цель программиста – оптимально использовать возможности платформы для выполнения этой задачи, в первую очередь путем создания адекватного пользовательского интерфейса.

Для создания приложений в MeeGo достаточного использования библиотеки *Qt*. Приложение будет вполне работоспособным, но его пользовательский интерфейс придется вручную адаптировать к особенностям MeeGo. Чтобы избежать такой адаптации, создана специализированная библиотека *Qt* – MeeGoTouch, – пользовательские интерфейсы в которой уже адаптированы к MeeGo.

9.3. Инструкция по выполнению лабораторной работы

Наша задача – научиться работать со специфическими элементами пользовательского интерфейса библиотеки MeeGoTouch. Для этого необходимы базовые знания языка C++ и знакомство с основами фреймворка *Qt*. В качестве целевого устройства мы использовали MeeGo 1.1.99 Tablet на планшете 3Q TU1102T, но также возможно использовать виртуальную машину или эмулятор.

Мы рассмотрим следующие элементы управления: страницы и списки. Для самостоятельно изучения можно порекомендовать разные виды кнопок (MButton), систему меню (MApplicationMenu), а также работу с панелью инструементов (tool bar).

Страница – контейнер для других элементов управления, аналог закладок в традиционных пользовательских интерфейсах. Ее появление вызвано тем, что на мобильных устройствах экраны небольшого размера, а элементы управления, наоборот, крупные, рассчитанные на работу с пальцами. На одном экране невозможно разместить большое количество элементов, поэтому приходится разносить их на разные страницы. Альтернативный подход – экран с прокруткой, который можно «сдвигать» в поисках нужного элементы, этот подход мы рассматривать не будем.

В MeeGoTouch страницы реализованы классом MApplicationPage, есть встроенная функциональность по

управлению их отображением. Для того, чтобы страница отобразилась на экране, достаточно вызвать для нее метод appear(), например, в обработчике нажатия на кнопку:

```
void MyApp::switchPage2() {
    page2.appear(&window);
}
…
connect(but,SIGNAL(clicked()),this,
SLOT(switchPage2()));
```

При этом одномоментно на экране может быть только одна страница. Программист сам решает, в какой момент это должно быть сделано, а система ведет учет открытых страниц и предоставляет пользователю кнопку «Back» для возврата.

Для навигации между страницами могут быть использованы различные схемы. Простейший пример приведен в этой лабораторной работе, когда переключение между страницами осуществляется нажатием кнопки на самой странице. Такой подход годится для каких-то вложенных настроек или в случае последовательной работы какого-либо мастера. Другой способ – разместить кнопки управления в области «Tap Bar», где они будут всегда доступны пользователю и он сможет вызвать желаемую страницу в любой момент. Минус в том, что навигационные кнопки постоянно занимают дефицитное место на экране. Еще один способ – использовать меню приложения для переключения страниц.

Один из наиболее сложных элементов управления – список. Во-первых, он предназначен для работы с большим количеством данных, во-вторых, списки в *Qt* традиционно сделаны довольно громоздко. В MeeGoTouch список реализован классом MList, в нем по умолчанию задействована только одна колонка, это вполне уместно для небольшого экрана мобильного устройства.

Для того, чтобы увидеть свой список на экране, программист должен реализовать как минимум два класса – класс с описанием данных для списка и класс, отвечающий за визуализацию ячеек. Описание этих классов желательно размещать в отдельном файле. Такая сложная процедура унаследована из *Qt* и кажется слишком тяжеловесной – в Android для создания списка достаточно указать ему массив строковых значений.

Класс с описанием данных наследуется от QAbstractListModel, в нем достаточно переопределить функцию rowCount(), которая возвращает количество строк в списке, и собственно функцию для данных:

```
QVariant ListModel::data(const QModelIndex &index,
int role) const {
  if (role == Qt::DisplayRole) {
      QStringList rowData;
      rowData << QString("Row %1").arg(index.row());
      rowData << "some details";
      return QVariant(rowData);
  }
  return QVariant();
}
```

Отметим, что для передачи данных используется QVariant – тип-контейнер, в который можно «положить» значения разных типов. В данном случае используется QStringList, но можно использовать и свою структуру данных – главное, чтобы это было согласовано с классом-визуализатором.

Визуализатор ячеек унаследован от класса MAbstractCellCreator<MContentItem>. Обратите внимание на нетипичное для Qt использование шаблона – этот класс не является наследником QObject! MContentItem – специальный виджет, предназначенный для отображения отдельной ячейки списка, именно он занимается отрисовкой упомянутых выше двух строк и картинки. При более глубокой работе со списком программист может заменить этот класс на свой собственный, более продвинутый. В простейшем же случае достаточно переопределить одну фунцию в MAbstractCellCreator<MContentItem>:

```
void CellCreator::updateCell(const QModelIndex&
index, MWidget *cell) const
{
    MContentItem *contentItem =
qobject_cast<MContentItem *>(cell);
    QVariant data = index.data(Qt::DisplayRole);
    QStringList rowData = data.value<QStringList>();
```

```
    contentItem->setTitle(rowData[0]);
    contentItem->setSubtitle(rowData[1]);
    contentItem-
>setImage(makeImage(50,50,rowData[0],8,Qt::red));
}
```

Как можно видеть, этот метод работает с тем самым QVariant, который генерирует предыдущая функция. Более правильным было бы упаковать в него и QImage, но мы этого не сделали для наглядности материала.

9.4. Задания для самостоятельной работы

1. Создать приложение для работы с двумя датчиками одновременно.
2. Создать *Qt*-приложение без использования MeeGoTouch, так чтобы его пользовательский интерфейс реагировал на датчик ориентации.

9.5. Выводы

В этой лабораторной работе рассмотрен пример рассылки SMS-сообщений по списку контактов.

9.6. Контрольные вопросы

1) Что такое User Experience?
   1. Тип приложения
   2. Оформление рабочего стола MeeGo
   3. Ощущения человека

2) Кого в коллективе разработчиков не касаются вопросы User Experience?
   1. Прикладного программиста
   2. Специалиста по usability
   3. Специалиста по human resource
   4. Маркетолога

3) Как соотносятся *Qt* и MeeGoTouch?
   1. MeeGoTouch построена на базе *Qt*
   2. *Qt* построена на базе MeeGoTouch
   3. Это две независимые библиотеки

4) Какую задачу решают страницы (MApplicationPage) в MeeGoTouch?
   1. Упрощают создание Интернет-приложений
   2. Позволяют разместить большое количество элементов на маленьком экране
   3. Облегчают ввод текста с экранной клавиатуры

5) Сколько страниц (MApplicationPage) могут быть доступны одновременно?
   1. Одна
   2. Сколько определит программист
   3. Сколько захочет пользователь

6) Какая навигация по страницам (MApplicationPage) встроена в MeeGoTouch?
   1. Кнопка «Вперед» и «Назад»
   2. Кнопка «Закрыть» и «Вперед»
   3. Кнопка «Назад»

7) Для чего необходим подкласс QAbstractListModel при работе со списком?
   1. Для указания геометрических размеров списка
   2. Для получения данных для списка
   3. Для совместимости с Qt3

8) С помощью какого типа данных передается содержимое ячейки списка?

1. QVariant
2. QString
3. void*

9) Для отображения какой информации предназначен виджет MContentItem ?
   1. Строка
   2. Изображение и строка
   3. Изображение и две строки

10) В каком случае испускается сигнал MList::itemLongTapped() ?
   1. Если элемент списка слишком длинный
   2. При длительном нажатии на элемент списка
   3. При перемещении элемента списка

Список литературы

1. MeeGo Touch Reference Documentation. http://apidocs.meego.com/1.1/platform/html/libmeegotouch/main.html
2. Handset UI Guidelines. https://meego.com/developers/ui-design-guidelines/handset

10. Лабораторная работа № 5 «Улучшение User Expirience/Usability с помощью gestures»

10.1. Цель лабораторной работы

Демонстрация процесса разработки практического приложения.

10.2. Инструкция по выполнению лабораторной работы

Задача видеонаблюдения естественным образом распадается на составные части. Во-первых, нам нужно уметь подключаться к Интернет с помощью GSM-модема. Во-вторых, мы должны научиться получать кадр от камеры и сохранять его в формате JPG. В-третьих, нам нужен механизм для приема данных на ЦОД. Эти подзадачи независимы друг от друга.

Основное устройство ввода на планшетном компьютере – сенсорный экран, он заменяет и «мышь», и клавиатуру. Однако его возможности гораздо шире, чем у «мыши», и их можно использовать для того, чтобы сделать интерфейс максимально удобным для пользователя. Одним из способов это сделать является создание и использование gestures (жестов).

Для этого нам будут необходимы базовые знания языка C++, знакомство с основами фреймворка Qt и библиотекой MeeGoTouch. Также необходим планшет или смартфон с сенсорным экраном, работающий под управлением MeeGo. Мы использовали MeeGo 1.1.99 Tablet на планшете 3Q TU1102T. На эмуляторе или виртуальной машине поддержки сенсорного экрана скорее всего не будет.

Gesture (жест) – фиксированная комбинация прикосновений пользователя к сенсорному экрану. Вообще говоря, для управления компьютером со стороны человека жесты могут быть использованы не только посредством сенсорного экрана. Сейчас существуют системы, способные выделять и опознавать обычные жесты человека с помощью видеокамер, разрабатывается «невидимые» клавиатуры, когда прикосновение к поверхности, например, стола в определенном месте воспринимается компьютером как нажатие соответствующей клавиши. Давно известны «жесты» курсором мышки на экране обычного компьютера. Далее в этой работе мы будем употреблять термин gesture только для работы с сенсорным экраном.

Gesture – это не просто движение пальца по сенсорному экрану, а движение в какой-то мере формализованное, которое может запомнить пользователь и распознать система. Благодаря этому у пользователя складываются образцы поведения, накапливается опыт общения с устройством. С этой точки зрения наиболее выигрышны gesture, которые встроены в систему изначально и используются многими программистами во многих приложения. В то же время для отдельных приложений программист вполне может захотеть использовать возможности сенсорного экрана по максимуму и создать свою собственную gesture.

Таблица 1.

Gesture	Действие пользователя	Класс *Qt*
Pan	Прикоснуться, перетащить, отпустить	QPanGesture
Pinch	Прикоснуться двумя пальцами, переместить их и отпусить	QPinchGesture
Swipe	Прикоснуться, быстро перетащить, отпустить	QSwipeGesture
Tap and hol	Прикоснуться и удерживать	QTapAndHoldGesture
Tap	Прикоснуться и отпустить	QTapGesture

В табл. 1 представлены все пять встроенных в *Qt* gestures, описание действий пользователя и классов *Qt* для них. Их также

можно использовать в MeeGoTouch. Не исключены ситуации, когда одно и то же движение пользователя воспринимается системой как несколько gestures одновременно. Некоторые виджеты MeeGoTouch имеют встроенную поддержку отдельных gestures, например, подклассы MPannableWidget или виджеты с контекстным меню.

Обрабатывать gestures можно как для отдельного виджета, так и для контейнера, такого, например, как страница приложения. Хотя для встроенных gestures выделены специальные обработчики, соответствующие события можно также перехватывать внутри event(QEvent *event) или gestureEvent(QGestureEvent *event). Проще всего, однако, просто переопределить обработчики для встроенных gestures.

Чтобы разрешить виджету получать информацию о gestures, необходимо вызвать для него метод grabGesture(Qt::SwipeGesture), указав желаемый тип. Для PinchGesture делается исключение – поскольку она использует возможности мультисенсорного экрана, ее обработка несколько сложнее и для нее также необходимо вызвать setAcceptTouchEvents(true). Это особенность реализации gestures в *Qt*.

Простейший пример обработчика встроенной gesture:

```
void MyGestPage::swipeGestureEvent(QGestureEvent
*event,
                                QSwipeGesture
*gesture) {
    if (gesture->horizontalDirection() ==
QSwipeGesture::Left) {
        if (id > 0)
            ((MyApp*)qApp)->switchPage(id-1);
    } else if (gesture->horizontalDirection() ==
QSwipeGesture::Right) {
        if (id < 2)
            ((MyApp*)qApp)->switchPage(id+1);
    }
}
```

Обратите внимание на избыточность параметров. Для получения необходимой информации достаточно и QEvent, который

надо будет привести к QGestureEvent и затем извлечь из него gesture, однако все это уже сделано для удобства прикладного программиста. Логика обработчика анализирует движения пользователя вдоль горизонтальной оси и в зависимости от направления движения переключает страницы приложения вперед или назад.

Направление движения привязано к физическому устройству, а не к его текущей ориентации в пространстве. То есть если перевернуть планшет «вверх ногами», то право и лево для пользователя и gesture перестанут совпадать, несмотря на то, что пользовательский интерфейс среагирует на датчик ориентации. Этот факт надо учитывать при программировании реальных приложений.

Более сложной задачей является создание собственной gesture. Она решается в три этапа – описание распознавателя для gesture, регистрация его в системе и обработка сообщений от него.

Самое сложное – создание распознавателя, он реализуется в подклассе QGestureRecognizer. Как минимум необходимо переопределить метод recognize():

```
Result MyGest::recognize(QGesture *gesture, QObject
*watched, QEvent *event) {
    if (event->type() == QEvent::TouchUpdate) {
        QTouchEvent *te =
static_cast<QTouchEvent*>(event);
        if (te->touchPoints().size() == 2) {
            gesture->setHotSpot(te-
>touchPoints().first().screenPos());
            return FinishGesture;
        }
    }
    return Ignore;
}
```

Приведенный пример – простейший, он считает gesture любое прикосновение к экрану двумя пальцами. К сожалению, сенсорный экран нашего планшета не поддерживает работу более двух пальцев одновременно. Более продвинутые алгоритмы могут учитывать координаты точек, другие сообщения от сенсорного экрана и т.п.

Тем не менее, отметим два момента. Во-первых, в случае успешного распознавания метод возвращает FinishGesture, если событие не имеет к методу отношения, он возвращает Ignore. Также есть и другие возвращаемые значения, они описаны в документации. Во-вторых, gesture необходимо установить HotSpot, чтобы система могла отправить ее соответствующему виджету, и он, в свою очередь, смог ее обработать. В более сложных случаях также переопределяются методы create() и reset().

Регистрация нового распознавателя осуществляется статическим методом QGestureRecognizer::registerRecognizer, который возвращает идентификатор типа новой gesture. С помощью флага Qt::CustomGesture можно определить, самодельный это тип или встроенный.

Затем этот идентификатор следует использовать в обработчике, чтобы выделить события, относящиеся именно к этой gesture (естественно, специализированные обработчики уже нельзя использовать).

```
void MyGestPage::gestureEvent(QGestureEvent *event) {
    QGesture *gest = event->gesture(myGestType);
    if (gest) {
        setTitle(QString("Got %1").arg(myGestType));
        return;
    }
    MApplicationPage::gestureEvent(event);
}
```

В данном обработчике у события запрашивается gesture желаемого типа, и если он есть – выполняется какое-то действие. В противном случае вызывается метод базового класса для обработки других gesture.

Если gesture перестал быть актуальным, от него можно избавиться методом unregisterRecognizer().

10.3. Задания для самостоятельной работы

1. Добавить в приложение слайдер или прогресс-бар и привязать его поведение к PinchGesture
2. Создать собственную gesture, которая будет распознавать начертание угловых скобок ('<' и '>') и организовать с ее помощью навигацию по страницам приложения.

10.4. Выводы

В этой лабораторной работе рассмотрен пример рассылки SMS-сообщений по списку контактов.

10.5. Контрольные вопросы

1) Что такое gestures ?
   1. Особый класс событий *Qt* GUI
   2. Комбинация прикосновений пользователя к сенсорному экрану
   3. Виджет, который управляется прикосновениями

2) Где описываются встроенные gestures ?
   1. В библиотеке *Qt*
   2. В библиотеке MeeGoTouch
   3. В библиотеке libgestures

3) Где возможна обработка событий gestures ?
   1. Только в страницах приложений
   2. Только в виджетах
   3. И в страницах приложений, и в виджетах

4) Что нужно сделать, чтобы обработывать SwipeGesture страницей ?

1. Достаточно переопределить обработчик
2. Необходимо переопределить обработчик и вызвать grabGesture()
3. Необходимо переопределить обработчик и вызвать grabGesture() и setAcceptTouchEvents()

5) Чем вызвана необходимость вызывать setAcceptTouchEvents() при работе с некоторыми gestures ?
   1. Спецификой работы *Qt* с мультисенсорным экраном
   2. Системными особенностями MeeGo
   3. Такой необходимости нет

6) Какое из перечисленных имен обозначает класс события gesture ?
   1. QSwipeGesture
   2. QSwipeGesture::Left
   3. QGestureEvent

7) Какие методы обязательно должны быть переопределены в классе QGestureRecognizer ?
   1. create()
   2. recognize()
   3. create() и recognize()

8) Что возвращает QGestureRecognizer::registerRecognizer()?
   1. Объект класса QGesture
   2. Обработчик для новой gesture
   3. Идентификатор типа новой gesture

9) С помощью какой битовой операции и флага Qt::CustomGesture можно проверить, что gesture создана прикладным программистом ?

1. И
2. ИЛИ
3. Исключающее ИЛИ

10) Какой обработчик нельзя использовать для созданных gesture ?
    1. gestureEvent()
    2. event()
    3. pinchGestureEvent()

Список литературы

1. Using Multi-Touch and Gestures with *Qt* http://www.slideshare.net/qtbynokia/using-multitouch-and-gestures-with-qt
2. Gestures and Multitouch
3. http://apidocs.meego.com/1.1/platform/html/libmeegotouch/gestures.html

Заключение

Основные итоги изучения курса и планы на будущее.

Уважаемые читатели и слушатели, вот и подошел к концу наш курс о том, как разрабатывать приложения на платформе Atom/MeeGo для планшетников и нетбуков. Несмотря на быструю изменчивость ИТ-технологий, мы надеемся на его полезность и актуальность. Все-таки основное внимание в курсе мы старались сфокусировать на общих проблемах и подходах к их решению. При всей этой общности мы использовали и конкретные конструкции и свойства новой платформы, рассмотренные примеры могут быть практически сразу использованы в практике разработки приложений для мобильных устройств.

Какова на наш взгляд дальнейшая судьба этого курса? Вам, наверное, интересно знать: будем ли мы его развивать, продолжать и углублять? Почему бы и нет. НО в ближайшей перспективе мы планируем разработать более социально направленный курс с условным название «Решение проблем с помощью вычислительных устройств», в котором в большей степени сконцентрироваться на постановках задач, на том, как и откуда они появляются, что сейчас и в ближайшем будущем ИТ-технологии могут дать для их решения.