Санкт-Петербургский государственный университет

Математико-механический факультет

Лаборатория Системного ПРограммирования и

ИНформационных Технологий (СПРИНТ)

Введение в разработку приложений на платформе Atom/MeeGo

О. Н. Граничин, В. И. Кияев, А. В. Корявко, С. А. Леви,

К. С. Амелин, Е. И. Антал, В. И. Васильев

Учебное пособие

Санкт-Петербург
2011

ББК __.__
Г __

Р е ц е н з е н т ы: д-р физ.-мат. наук, профессор А.Н. Терехов
(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 и я е в В. И., К о р я в к о А. В. и др.
Г __ Введение в разработку приложений на платформе Atom/MeeGo. Учебное пособие. – СПб., 2011. – 293с.

ISBN

В учебном пособии представлены материалы одноименного курса по разработке приложений для мобильных устройств, прочитанного авторами для стажеров Лаборатории системного программирования и информационных технологий (СПРИНТ, СПбГУ-Intel). Рассматриваются как общие основы разработки приложений для мобильных устройств, так и специальные инструменты, ориентированные на новую операционную систему MeeGo и процессоры Intel Atom. Рассказано также об основных принципах реализации качества программного приложения (продукта) и о путях его практической коммерциализации.

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

Библиография – 107 назв., таблиц – 12, рисунков – 77.

ББК __.__

ISBN

Содержание

1. Введение

Мобильность и супервычисления. Структура и авторы курса.

Наиболее характерная черта нынешнего развития мировой экономики — колоссальные успехи и достижения в области техники и технологии, развитие наукоемких производств. Высокие темпы развития науки и технологий, а главное, масштабы и темпы их внедрения в производство и общественную жизнь, превратили научно-техническую революцию в естественный процесс, она стала перманентной. Благодаря всепланетному масштабу коммуникаций и росту уровня образования «ноу-хау» сейчас практически сразу после изобретения становятся общечеловеческим достоянием. В условиях динамичного развития рынка, усложнения его инфраструктуры информация становится таким же стратегическим ресурсом, как и традиционные материальные и энергетические. Современные технологии, позволяющие создавать, хранить, перерабатывать и обеспечивать эффективные способы представления информации, стали важным фактором конкурентоспособности и средством повышения эффективности управления всеми сферами общественной жизнедеятельности. Уровень информатизации является сегодня одним из главных факторов успешного развития всякого проекта, предприятия и организации.

Молодое поколение уже родилось в эпоху компьютеров, телевизоров, мобильных телефонов, Интернета и т. п. В разговорах со студентами преподаватели старшего поколения часто ловят себя на мысли, что для большинства из них компьютер - это привычная вещь, от которой не ждут ничего необычного!

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»? «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. Среди новых характерных черт будущей парадигмы все более отчетливо проступают следующие: процесс вычисления ─ набор одновременно работающих асинхронных моделей взаимодействующих динамических систем; стохастичность; гибридность.

Каковы тенденции изменений в организации взаимодействий с суперкомпьютерами? Сегодня положение во многом напоминает школьный опыт одного из авторов, относящийся к концу 70-х годов прошлого века, когда разработчик записывал на специальном бланке текст программы, потом кто-то подготавливал программу к загрузке в ЭВМ («набивал» перфокарты), потом, сдав пакет (колоду перфокарт) на счет, разработчик несколько дней ждал ответа, после получения разбирался с результатом и т. п. Сейчас также в полный рост стоит проблема организации эффективного взаимодействия с суперкомпьютером. Время сбора и подготовки данных становится критичным.

В мире ИТ уже во многих областях побеждает другая схема, в которой нет четкого выделения трех изолированных стадий: подготовка данных, счет, анализ результатов. Этому способствовало всеобщее проникновение персональных компьютеров, смартфонов, мобильных телефонов, которые кардинально изменили представления о месте ЭВМ в обычной жизни. Компьютеры уже интегрированы во многие сферы человеческой деятельности, разработчики встраивают миниатюрные вычислительные устройства и специализированные программы во внутрь многих процессов. В массовом сознании о роли ЭВМ в практической жизни уже произошел сдвиг от супервычислителя к мобильности, понимаемой в очень широком смысле.

Противоречивость мобильности и супервычислений сейчас имеет объективные основы (разные аппаратные платформы, назначения устройств и т. п.), но не слишком ли сильно в умах людей эти аспекты противопоставляются? Посмотрим на перспективы развития супервычислений. При движении к ExaScale переход к новой парадигме вычислений приведет, наверное, к тому, что архитектура вычислительных устройств «сдвинется» в сторону, как было сказано выше, «*набора одновременно работающих асинхронных моделей взаимодействующих динамических систем*». На это можно смотреть как на своеобразную «внутреннюю» мобильность суперкомпьютера. С другой стороны, уже сейчас работа многих мобильных сервисов, связанных с решениями научных, прикладных и управленческих задач, обеспечивается «в облаках». За внешней простотой общения для пользователей в основе «облачной» технологии скрываются мощные Data-центры и суперкомпьютеры!

А нужно ли противопоставлять мобильность суперкомпьютингу и воплощать в компьютерную жизнь принцип «миллионы муравьёв победят слона». Супервычислениям в будущем нет другой альтернативы, как широко использовать мобильные технологии. На смену жесткому разделению процессов:

<Вход – Счет – Выход>,

придет концепция с «размытыми» контурами.

В действительности мобильность и высокопроизводительные вычисления, по нашему мнению, будут всё более тесно связаны. Посмотрим на пример такой традиционной области для ИТ, как «кодирование/декодирование информации». Полвека развития со времен пионерских работ Котельникова-Найквиста-Шеннона успели убедить несколько поколений в незыблемости и правильности постулатов о минимальном числе отсчетов необходимых для точного восстановления сигнала (функции). Но развитие технологий в 90-е годы прошлого века позволило массово перейти от обработки одномерных сигналов к изображениям, а сейчас на повестке дня стоит решение задач трехмерного телевидения. Однако реальные задачи обработки таких сигналов для традиционных средств кодирования/декодирования уже становятся «не по плечу» даже суперкомпьютерам!

Что же происходит сейчас у нас на глазах с развитием и изменением «парадигмы кодирования/декодирования»? Вместо классической появляется новая парадигма «Compressive Sensing», для которой еще даже не придумано русское название. В чем ее суть? Пользователям в реальном времени обычно не нужна вся информация о процессах в каждый момент времени, а важно только изменение принципиальных компонент. Например, при передаче ведеопотока с трехмерной сцены достаточно однократно передать общий вид и детали, выделить характерных персонажей, а потом только отслеживать их изменения. Оказывается, имея интеллектуальные сенсоры (использование принципа мобильности сбора информации), целесообразно до момента кодировки и передачи данных вместо определения всех значений трехмерной сцены (обычно это огромное число) сначала произвести несколько «сверток» всей сцены с некоторыми рандомизированными матрицами, а потом закодировать и передать для обработки суперкомпьютеру только эти несколько измерений. А это означает, что уже сейчас можно говорить о том, что мобильность и использование

интеллектуальных агентов при сборе и предварительной обработке данных может существенно сократить время подготовки данных для суперкомпьютерных вычислений. Если глубже взглянуть на новую парадигму кодирования/декодирования, то становится ясным, что при ней существенно меняется не только процесс подготовки данных, но и сам процесс обработки.

Воплощение новой парадигмы суперкомпьютерных вычислений невозможно в отрыве от подготовки новых кадров. Это очень важно при обучении студентов. Они не должны «снижать горизонт» своих амбиций, должны понимать и верить, что и на их долю осталось много важных проблем, не имеющих пока приемлемых решений. В Санкт-Петербургском университете при поддержке корпорации Intel организована Учебно-исследовательская лаборатория Системного ПРограммирования и ИНформационных Технологий (СПРИНТ, СПбГУ-Intel). Если зайти на сайт Лаборатории СПРИНТ http://sprint-intel.phys.spbu.ru/, то в глаза бросаются два мощных крыла:

- *технологии высокопроизводительных вычислений и*
- *«мобильные» технологии.*

Сегодня эти технологии разделены классической парадигмой, формой и содержанием задач, областью применения результатов. Их объединение произойдет само собой, когда абстрактные рассуждения будут реализованы на практике.

Это можно показать на реальном примере исследовательского проекта «Мультиагентное взаимодействие группы легких беспилотных летательных аппаратов (БПЛА)». Трудно переоценить актуальность такой задачи для России с её огромной территорией, сельскохозяйственными угодьями, лесами, множеством газо- и нефтепроводов, ЛЭП и «пожарами». При создании такой системы как раз и проявляются все обсуждавшиеся выше проблемы. Многие из задач – разведка полезных ископаемых, поиск людей, выяснение причин техногенных катастроф, плановое патрулирование территорий и т.п. требуют для получения практических результатов применения супервычислений с огромным количеством исходных данных, которые должны поступать на суперкомпьютер в режиме реального времени. Если используются традиционные БПЛА с оператором и приемом/передачей данных по радиоканалу, то реальные автоматизированные системы «захлебываются» от потоков данных, которые надо еще сохранять и передавать компьютеру для обработки. Об обратном потоке управления БПЛА обычно даже речь и не идет, так как результаты не удается получить в реальном времени.

Итак, сегодня в развитии информационных технологий на передний план выдвигается мобильность. Вместе с развитием высокопроизводительных вычислений, суперкомпьютеров, центров обработки данных, переходе к парадигме «вычислений в облаках» меняется место и роль мобильных устройств. От переносных телефонов и пультов дистанционного управления мы шагнули не просто к точкам входа в порталы разнообразных сетей и облаков а сейчас мобильные устройства – это высокоинтеллектуальные точки входа, которые способны решать большой круг задач автономно, самостоятельно получая ответы или предварительно обрабатывая входную информацию для более эффективной передачи и дальнейшей обработке, или существенно преобразовывая (визуализируя) результаты с серверов и облаков.

Основу пособия составил курс лекций по разработке приложений для мобильных устройств, прочитанный авторами осенью-зимой 2010 года для стажеров Лаборатория системного программирования и информационных технологий СПбГУ, созданной и финансируемой при поддержке корпорации Интел.

Главная цель пособия — дать общее представление о процессе разработки приложений для мобильных устройств, функционирующих под управлением операционной системы Linux, а также о специальных инструментах, ориентированных на новую операционную систему MeeGo и процессоры Atom-Intel. Настоящее пособие следует рассматривать в качестве общего вводного курса.

Курс состоит из трех разделов. Первый носит общий характер: сведения о платформе Atom/MeeGo, о том как разрабатывать приложения, о том, чем пользоваться и как все это может использоваться. Примеры разработки приложений собраны в последней части вместе с общими вопросами качества разработки приложений и возможностей их коммерциализации. Середина курса фокусируется на библиотеках MeeGo, на описании приемов того, как что-то можно сделать, используя новые перспективы и возможности Atom/MeeGo. Параллельно с лекционным курсом

составлены лабораторные занятия основная цель которых – научить как на такой легкой и общедоступной платформе как Atom/MeeGo разрабатывать реальные приложения, например, разработать систему управления легким беспилотным летательным аппаратом (БПЛА).

Пособие состоит из введения, трех разделов и заключения, разделенных на 15 лекций, из которых 12 дополнительно снабжены лабораторными работами. Общий план и структуру курса разработали О.Н. Граничин, В.И. Кияев, А. В. Корявко и С. А. Леви при активном участии И. О. Одинцова. Лекции 1,2 и 15 составлены О.Н. Граничиным, 3,12 – К.С. Амелиным, 4-8 – С.А. Леви совместно с Е.И. Антал, 9-11 – В. И. Васильевым, 13-14 – В. И. Кияевым. Лабораторные работы 1, 12 составлены К. С. Амелиным, 2-6 – С. А. Леви, 7 – В. И. Васильевым, 8-11 – А. В. Корявко. Учебные материалы доступны на сайте лаборатории СПРИНТ по ссылке http://sprint-intel.phys.spbu.ru/ index.php?option=com_content&view= category&layout=blog&id=14&Itemid=31.

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, за постоянное внимание к продвижению проекта, многочисленные предложения и замечания, а также еще одного рецензента – заведующего кафедрой системного программирования математико-механического факультета СПбГУ профессора Андрея Николаевича Терехова, сделавшего ряд ценных замечаний, которые были использованы при подготовке пособия к печати.

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

Список литературы

1. Граничин О.Н., Кияев В.И. Информационные технологии в управлении. --- М.: Изд-во Бином. 2008. 336 с.
2. Сб. тр. Всерос. научн. конф. «Научный сервис в сети Интернет: суперкомпьютерные центры и задачи» (20-25 сентября 2010 г., г. Новороссийск). М. 2010.
3. Амелин К.С., Граничин О.Н., Кияев В.И. Суперкомпьютеры или мобильность: кто кого? // Суперкомпьютеры, 2010, № 4, с. 30-33.
4. Граничин О.Н., Павленко Д.В. Рандомизация получения данных и l_1-оптимизация // Автоматика и телемеханика, 2010, № 11, с. 3 -28.

Раздел 1. Общие сведения о платформе Atom/MeeGo и разработке приложений

2. Общие сведения о платформе Atom/MeeGo

К системам на кристалле (SoC) – закономерная тенденция развития. Характеристики и возможности процессора Intel Atom, перспективы использования. ОС MeeGo – цели создания и общая характеристика, архитектура.

Олимпийские спортивные лозунги: “Быстрее, Выше, Сильнее”, – в компьютерной области трансформировались в: “Быстрее, Миниатюрнее, Больше памяти”. На самом деле эти противоречивые цели стыкуются в одном: технологии приближаются к созданию искусственного интеллекта (конечно же, не завтра), уже скоро мобильные вычислительные устройства смогут выполнять те функции, которые несколько лет назад и «не снились» пользователям компьютеров.

2.1. К системам на чипе – современная тенденция развития вычислительной техники

Основная тенденция развития средств вычислительной техники – миниатюризация. 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, к данным можно будет получать параллельный одновременный доступ разным вычислительным устройствам, для общения вычислительных устройств между собой будет общая 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 в стиль программирования, в то, как будут использоваться вычислительные устройства.

Ключевым моментом всех этих изменений является еще и мобильность.

Все более хрупкие и миниатюрные вычислительные устройства становятся и более мобильными. При этом они, конечно же, и не очень хрупкие. Речь идет не только о телефонах и нетбуках, серьезное внимание уделяется широкому спектру встроенных системам, которые могут включаться в разнообразные технологические процессы. О них как раз и будет весь этот курс, о том, как писать для них программы и чем для этого удобно пользоваться.

В чем видится новый «скачок» в развитии, который сейчас делает корпорация Интел и все сообщество разработчиков и производителей вычислительных систем?

Взглянем на историю развития средств вычислительной техники. За шесть десятилетий пройден путь от ламп, через транзисторы, интегральные микросхемы к сверхбольшим интегральным микросхемам (СБИС) (см. Рис. 2.1.1).

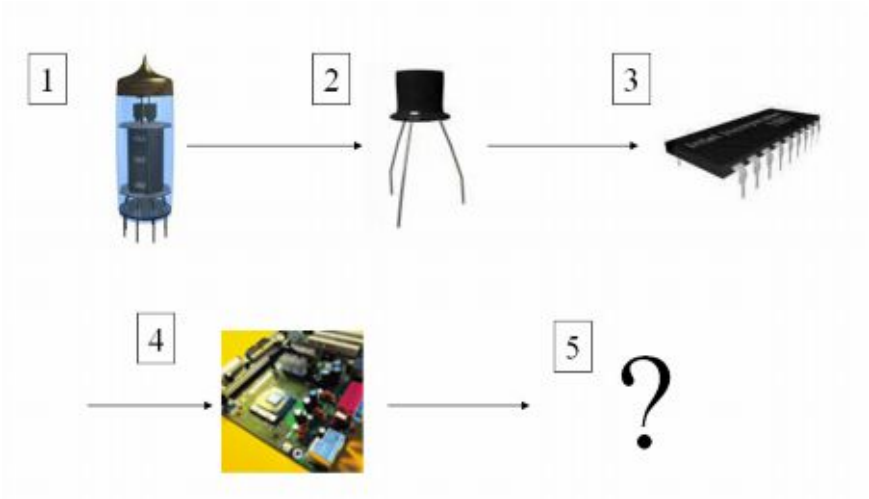


Рис. 2.1.1. Поколения компьютеров.

Что будет дальше? Основной вопрос – как обрабатывать данные?

Человечество накопило большой массив задач, которые компьютеры умеют эффективно решать. Можно послать ракету в определенную точку, можно быстро вычислить преобразование Фурье, можно быстро найти решение какого-нибудь важного уравнения и много-много других (см. Рис. 2.1.2). Но как нам поступить, если мы хотим собрать универсальный вычислитель, который способен один выполнить их все? Предположим, что мы делаем не вычислитель, быстро решающий уравнения, а наша цель сделать что-то похожее на нас, на людей (искусственное мыслящее существо), которое в условиях неопределенностей способно распознать реальную ситуацию, выбрать адекватную ей задачу и решить ее. Например, среди всего реализованного набора задач принимает решение о выборе блока, ответственного за решение определенной задачи, который «говорит: Да, эту ситуацию я контролирую, это моя цель, пора стрелять или вижу угрозу, пора убегать и т. п.»

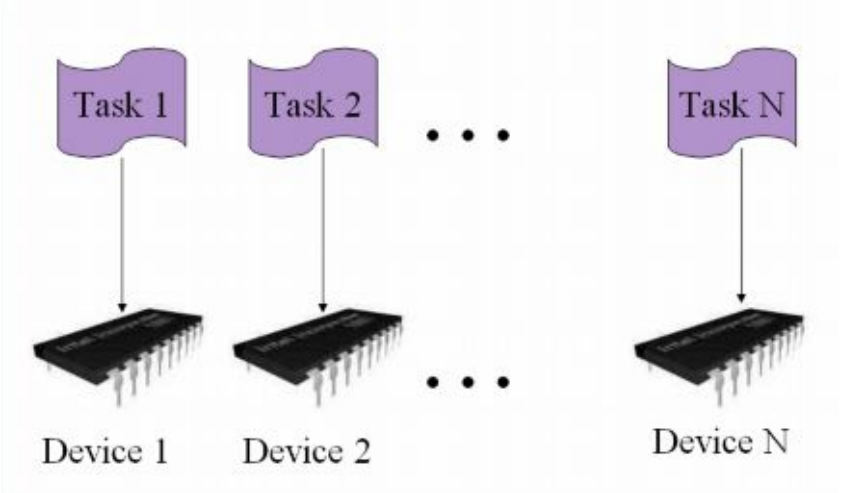


Рис. 2.1.2. Набор задач и решений.

Такие системы плохо «вписываются» в традиционную концепцию архитектуры компьютера, в которой операции обычно выполняются последовательно, данные загружаются последовательно, для выполнении того или иного действия надо последовательно пройти некоторые шаги А, Б, В и т. д., как-то их перебрать. Но пока мы их перебираем, зачастую решаемая задача перестает быть актуальной.

Как будет в перспективе? Простое решение собрать все вычислительные блоки вместе в сегодняшних условиях наткнется на проблемы отвода тепла, одновременной доставки информации, выбора ведущего блока и многими другими. Наверное, когда-то мы сможем собрать блоки

(микросхемы), решающие выбранные нами задачи (желательно все) в «клубок» закрученной спирали, как молекулы ДНК у нас в клетках. Наши молекулы ДНК в клетках решают огромное количество функций.

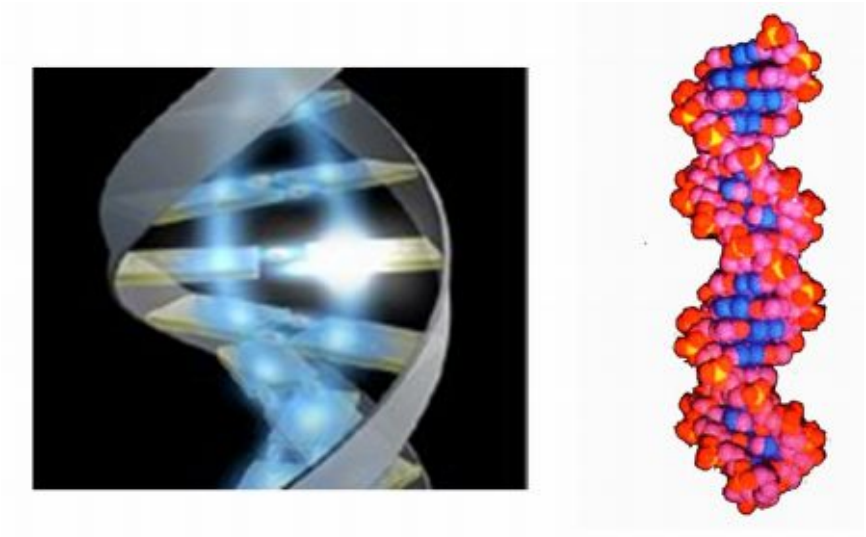
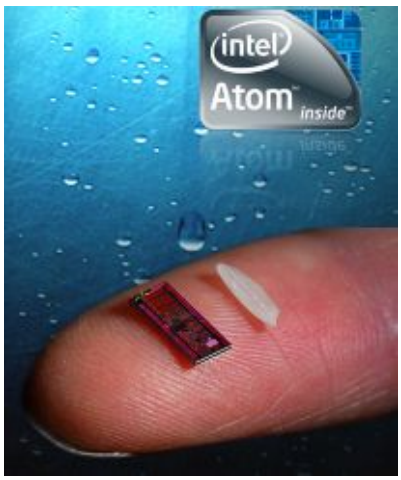


Рис. 2.1.3. Информационный резонанс (слева). Фрагмент молекулы ДНК (справа).

Рассмотрим пример, как могут решаться важные для универсального вычислителя задачи с доступом к памяти, с параллелизмом, данными. Гипотетически можно представить, что химическое или электромагнитное воздействие (например, луч света) на такой «клубок» может одновременно воздействовать сразу на все блоки и каждый из них одновременно с другими «примеряет» поступающую информацию «на себя». Традиционная альтернатива параллелизму – перебирать по очереди и смотреть кому лучше подходит. В том блоке, который распознал адекватность текущей информации его задаче, которому информация подошла лучше остальных, можно представить себе состояние некоторого *информационного резонанса* (см. Рис. 2.1.3).

Представленные рассуждения подтверждают, что соединение в одном месте (в одном чипе) нескольких задач – это общая тенденция развития вычислительной техники. Много-много разных задач в недалеком будущем будут решаться в одном чипе, как-то эти задачи и их решения будут собираться вместе, будут как-то решаться вопросы приоритетов, распределения заданий и т. п.

2.2. Процессор Intel Atom



Что есть уже сейчас? Конечно до чего-то похожего на Рис. 2.1.3 с ДНК еще далеко, но у корпорации Intel уже появился процессор Atom. Это самый маленький процессор Intel, он построен на самых маленьких транзисторах в мире. У него очень низкое энергопотребление, размером с рисинку! Но – это полноценная архитектура x86.

Где сейчас уже используются, или начнут использоваться в ближайшее время процессоры Intel-Atom? Это – хорошо знакомые всем персональные компьютеры, нетбуки/неттопы, мобильные Интернет-устройства, бытовая электроника с подключением к Интернет (например, IPTV), встраиваемые устройства (особенно для автомобилей разнообразные системы поддержки движения). Многие автомобилисты уже не представляют себе, как ездить на машине без навигатора.

Что собой представляют SoC (системы-на-чипе), которые уже сейчас производит корпорация Intel? Выше речь шла о будущих перспективах одновременного выполнения тысяч задач. Сейчас, как минимум, речь идет о переносе на чип контроллеров ввода-вывода, систем обработки видео и ряда других, существенно сокращая объемы вспомогательных обменов в памяти, повышая эффективность, производительность и упрощая конструкции. 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.

Что сейчас предлагает Intel на рынке процессоров для мобильных устройств и SoC (System on Chip, система-на-чипе).

Базовая платформа 2008 года – Menlow (Silverthorne Processor and Poulsbo System Controller Hub) – разработана по технологии 45 нм, размеры: 8500 кв мм, CPU/CS: 667 кв мм, Part Count: 550, мощность 1.4W 2.4-4.3W active, 1X graphics performance, 4-6 hrs video/browsing.

Следующая платформа 2009 года – Moorestown (Silverthorne Lincroft Processor, Langwell Platform Controller Hub, and Briertown MSIC) – разработана также по технологии 45 нм, размеры с кредитную карточку: 4200 кв мм, CPU+PCH: 387 кв мм, Part Count: 400, мощность >50x Lower 2-4X reduction in active, 2X graphics performance, Longer life with smaller battery, Enabling smaller FF.

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.

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 запуске 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*.

2.3. ОС MeeGo

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.

Сегодня уже огромное количество устройств работает на основе процессора Intel-Atom (см. Рис. 2.3.1).

С какой основной проблемой сталкиваются сегодня разработчики и пользователи? Все устройства на Рис. 2.3.1 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 какого-то 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. 2.3.1. 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.

Что произошло весной 2010 года? 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.

Как с практической стороны подойти к освоению нашего курса. Для начала надо установить MeeGo на свой нетбук или компьютер (см. инструкции к лаб. работе № 1). Потом надо будет установить на свой нетбук или компьютер «Среду разработки» (см. лаб. работу № ?). В последующих лекциях и лабораторных работах читатели найдут ответы на вопросы «Как себя одеть/обуть?». Потом – простые примеры, далее – разработка законченных полнофункциональных приложений.

Типичный вид экрана пользователя ОС MeeGo показан на Рис. 2.3.2. Ключевой является верхняя строка меню. Сразу у вас уже работает Интернет-браузер, календарь, почта, можно просмотреть и настроить свои подключения к сети и т. п.

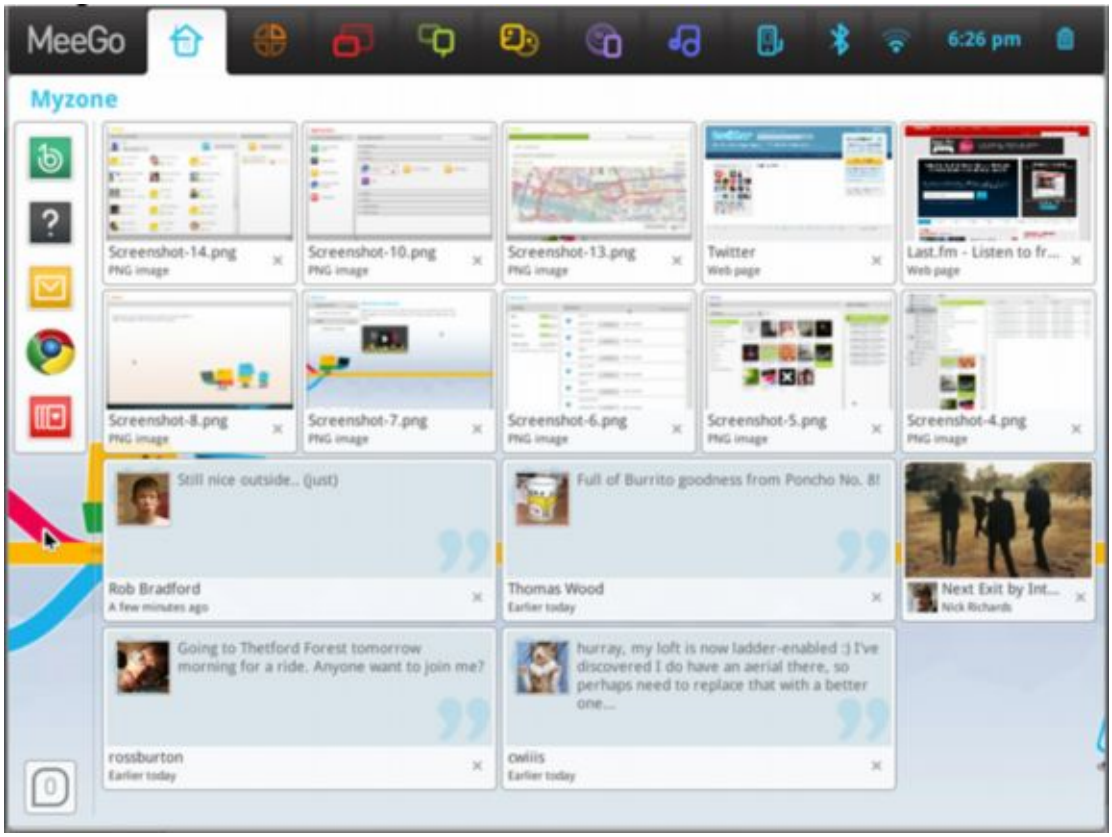


Рис. 2.3.2. Экран пользователя ОС MeeGo.

Структура следующих лекций и лабораторных работ во много соответствует архитектуре ОС MeeGo (см. Рис. 2.3.3). Архитектура ОС MeeGo (типичная и для многих других операционных систем) разделяется на три основных слоя: основа, средний и пользовательский.

В основе лежит ядро Linux с базой данных настроек, системными библиотеками, шиной сообщений и информацией о платформе. 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.

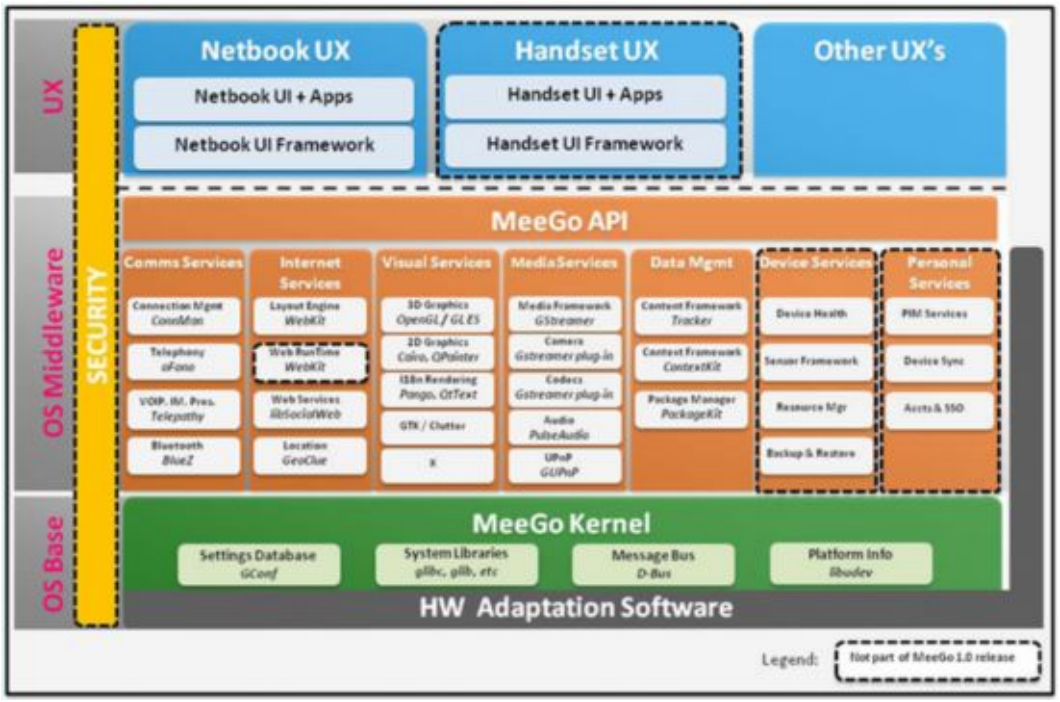


Рис. 2.3.3. Архитектура ОС MeeGo.

2.4. Выводы

Основная цель этой лекции – агитационно-разъяснитель-ная. 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.

2.5. 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) На телефоны какой из перечисленных фирм-производителей сейчас можно установить ОС MeeGo?
   1. Samsung.
   2. Sony-Ericson.
   3. Nokia.
   4. HP.
4) MeeGo – это проект:
   1. Apple.

2. Microsoft.
3. Соединение многих проектов.
4. Новый российский проект.

5) Для каких устройств предназначена ОС MeeGo?
   1. Для серверов.
   2. Для мобильных информационно-развлекательных.
   3. Для хранилищ данных.
   4. Для охранных систем банков.
6) Архитектура ОС MeeGo включает основных слоёв:
   1. Один
   2. Три
   3. Четыре
   4. Шесть
7) 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., Кияев В.И. Информационные технологии в управлении. --- М.: Изд-во Бином. 2008. 336 с.
2. Граничин О.Н., Молодцов С.Л. Создание гибридных сверхбыстрых компьютеров и системное программирование. СПб, 2006. 108с.
3. Ibrahim Haddad Introduction to the MeeGo Project. [On-line] http://wiki.meego.com/images/MeeGo_Introduction.pdf
4. http://www.intel.com/cd/corporate/pressroom/emea/rus/457838.htm

3. Основные аппаратные сегменты на платформе Atom/MeeGo

Нетбуки, хэндсеты, мобильные Интернет-устройства (MIDs, mobile internet devices), смарт-ТВ, системы поддержки в транспортных средствах (IVI, in-vehicle infotainment systems) и др.

3.1. Оборудование под MeeGo

В современном мире почти у каждого человека есть одно или несколько портативных устройств. Это нетбуки, коммуникаторы, информационно-развлекательные автомобильные устройства, портативные компьютеры (см. Рис. 3.1.1). Также всё чаще внедряются микрокомпьютеры в привычные для нас вещи. Появляются умные телевизоры, видеофоны, встроенные в авто системы. Именно для таких портативных устройств и встроенных систем предназначена ОС MeeGo.

Рис 3.1.1. Аппаратные сегменты на платформе Atom/MeeGo.

Какими же характеристиками и особенностью архитектуры должны обладать портативные устройства и встроенные системы для того, чтобы на них работала ОС MeeGo? Разработчиками выделен ряд основных критериев:

- 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.

Это основные критерии, которым должно обладать портативное устройство для работы с существующими образами MeeGo. Разработчиками приведён список поддерживаемого оборудования на официальном сайте www.meego.com, в этом списке приведены устройства построенные на процессоре Intel Atom и один из коммуникаторов Nokia N900 построенный на процессоре ARM Cortex-A8. На портале wiki.meego.com можно найти расширенный список поддерживаемых

устройств. Участники портала делятся своим опытом по установке ОС MeeGo на различные портативные устройства.

3.2. Адаптация MeeGo под новое оборудование

Операционная система MeeGo пока ещё работает не со всеми портативными устройствам и подключаемым к ним оборудованием. Число поддерживаемого оборудование постоянно растёт. Также разработчиками не исключается вариант адаптации MeeGo под конкретное оборудование. Для адаптации MeeGo необходимо отправить разработчикам пакет программных компонент, такие как:

1) Компоненты ядра.
   a) Драйверы. Сюда надо включить набор аппаратных драйверов необходимых для работы с устройством и подключаемым к нему оборудованию.
   b) Архитектура ядра. Пакеты отражающие особенности архитектуры ядра. Например, прошивки.
   c) Файл конфигурация ядра. Этот файл необходим для отладки оборудования. Он отражает необходимой детали конкретной конфигурации, которые необходимо для дополнения к файлу конфигурации MeeGo. Например, тип процессора, параметры отладки ядра, архитектура устройства и т.д.
2) Компоненты X Windows (графического интерфейса).
   a) Архитектура ядра. Дополнительные пакеты, которые отражают особенности архитектуры, и будут добавлены в графический интерфейс MeeGo. Например, тип графического контроллера, дисплея, т.п.
   b) Файл конфигурации графического интерфейса. Этот файл содержит параметры настройки для работы X Windows. Например, разрешение экрана и др.
3) Дополнительные компоненты и инструменты. Сюда необходимо включить загрузчик, инструменты для создания образа, параметры GSM модема (для устройств со встроенным или подключаемым GSM модемом), видео и аудио кодеки и другую дополнительную информацию по конкретному.

Далее эти компоненты рассматриваются “ведущим по пакету”, проверяется на полноту и достаточность информации об устройстве. Если информации достаточно, то ведущий по пакету отправляет всю информацию ведущему по проекту. Последний строит новую систему и собирает новый образ. Если информация об устройстве недостаточная, то ведущий по пакету не отправляет его дальше пока не будет вся информация.

3.3. Существующие образы MeeGo

Сейчас доступны четыре образа ОС MeeGo для портативных устройств:

- MeeGo для нетбуков
- MeeGo для нетбуков с Google Crome
- MeeGo для каммуникаторов
- MeeGo для информационно – развлекательных систем в авто.

Все эти образы построены на одном и том же ядре и среднем слое ОС MeeGo (смотри пункт 2.3.) Отличия заключается в верхнем *слое. Все эти образы можно скачать в свободном доступе с сайта* www.meego.com.

MeeGo для нетбуков

- Удобный и быстрый доступ к приложениям календарь, задачи, встречи.
- В режиме реального времени происходит обновления социальных сетей.
- Уже установленный Интернет-браузер Google Chrome или Chromium.
- Высоко оптимизированные под быстроту и производительность нетбука.

- Установленные приложения для просмотра текстовых файлов, аудио и видео файлов, работы с изображением.
- Языки: японский, корейский, китайский упрощенный, китайский традиционный, шведский, польский, финский, итальянский, бразильский португальский, французский, немецкий, испанский, русский, голландский, английский и британский английский.

MeeGo для коммуникаторов

- Упрощённый графический интерфейс, развиты приложения для работы с сотовой связью, для голосовых вызовов, SMS-сообщениями, просмотра веб-страниц, музыки и воспроизведения видео.

MeeGo для информационно – развлекательных автомобильных систем

- Удобная панель задач, разработанная для автомобильного центра консоли. По умолчанию, она появляется на левой стороне экрана, но легко можно переместить его в правую часть экрана, чтобы оптимизировать доступ для водителя или пассажира, по желанию.
- Поддерживается функция TTS (Text-To-Speech) для голосовых команд. TTS включена по умолчанию в ivihome меню навигации.
- Голосовые команды для ivihome были предварительно определены для навигации прокрутки меню.
- Встроенный приложения: видео-плеер, аудио-плеер, просмотр фотографий, handsfree и др.

3.4. Лабораторная работа № 1 «Установка ОС MeeGo на нетбук»

3.4.1. Цель лабораторной работы

Научиться записывать образ MeeGo ОС на Flash носитель с помощью приложений под ОС Windows и Linux, инсталлировать и запускать ОС MeeGo на нетбуке. Общее знакомство с ОС MeeGo.

3.4.2. Введение

Образ MeeGo предназначен для загрузки ОС и работы с USB Flash носителя. Netbook или PC может загружаться непосредственно с диска USB, не изменяя установленную ОС. Это позволяет тестировать ОС MeeGo перед установкой. Диск USB с образом, также можно использовать для установки MeeGo ОС на ваш Netbook или PC.

3.4.3. Инструкция по выполнению лабораторной работы

- Включите PC и отформатируйте USB Flash диск в формате FAT 32. Объем диска должен быть не менее 1 Gb.

Помните, что при форматировании вся информация, записанная на USB Flash носителе будет удалена!

- Найдите в Интернете и скачайте файл образа .img для нетбука (MeeGo for Netbooks) с ресурса http://meego.com/downloads/releases/netbook,
  на котором ресурсе предложено два варианта образа. Образ MeeGo v1.0 for Netbooks (Google Chrome Browser) и MeeGo v1.0 for Netbooks. Эти два образа отличаются встроенным браузером Google Chrome или Chromium соответственно. В случае установки первого потребует регистрацию браузера.
- Записать образ на USB Flash.

  Запись образа в ОС Windows

  - Скачать программу Win32DiskImager.exe, перейдя по ссылке: https://launchpad.net/win32-image-writer/+download(zip file)

- Разархивировать Win32DiskImager-RELEASE-0.2-r23-win32.zip в корень диска, на котором установлена ОС Windows.
- Открыть файл W32DiskImager.exe.
- Записать образ MeeGo в папку C:\win32diskimager-RELEASE-0.2-r23-win32.
- Прописать путь к .img и USB Flash диску.
- Записать образ на USB Flash, нажав кнопку "Write".

При записи образа может появиться сообщение об ошибке следующего содержания:

"An error occurred when attempting to get a handle on the device. Error 8:"

Варианты решения:

- Не отформатирован USB Flash диск (надо отформатировать в Fat 32).
- Программа Win32: image writer должна быть открыта от имени администратора.
- Слишком длинное название образа. (Максимально сократить название образа, например, просто meego).
- Слишком длинный путь к образу (запишите образ в папку программы win32: image writer, затем всю папку в корень жёсткого диска).

Запись образа в ОС Linux

- Убедитесь, что на диске Flash USB не записан другой образ, диск USB должен быть отформатирован.

```
# umount <usb-drive>
```

Можно использовать один из двух методов:

a) **Image Writer** (рекомендуется, требует Python>= 2.4)

Image Writer является небольшой Python исполняемый скрипт, который обнаруживает USB-диск и пишет образа. Преимущество использования изображений писателя является то, что он не будет непреднамеренно перезаписать системы жесткого диска.

- Скачать Image Writer. Ресурс для скачивания http://meego.com/downloads/releases/netbook

```
# cd <Директория, в которую записан образ>
# chmod a+x ./image-writer
# ./image-writer <Имя образа>
```

b) **Используйте 'dd' из командной строки**

Внимание: Запомните название вашего USB диска прежде чем продолжить!

```
# dd bs=4096 if=<Имя образа> of=<usb drive>
```

- Вставить Flash-диск в Netbook и запустить его с Flash носителя. Зайти в настройки BIOS и настроить приоритет запуска (boot) с Flash носителя. Сохранить изменения в Bios и перезагрузить PC. При загрузке с USB Flash появится меню, в котором представлено 3 варианта загрузки на выбор: Загрузка MeeGo с USB Flash, установка MeeGo на жесткий диск и загрузка с вашего жесткого диска.
- Просмотреть все возможные пункты меню и ознакомиться с их функциональностью.
- Запустить Интернет-браузер.

3.4.4. Задания для самостоятельной работы

1. Настройте приложение «Календарь» и введите в него план-график изучения курса.
2. Настройте приложение «Электронная Почта».
3. Если Вам не удалось автоматически подключиться к сети Wi-Fi, то скачайте необходимые драйверы и установите их.
4. Установите обновления для MeeGo.

3.5. Выводы

Материал этой лекция вместе с лабораторной работой № 1 поможет читателям и слушателям выбрать для себя необходимое устройство на платформе Atom/MeeGo, загрузить и проинсталлировать операционную систему и начать с ней работать.

3.6. Контрольные вопросы

1) 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.
2) Для какого устройства был выпущен первый релиз ОС MeeGo?
   1. Для нетбуков.
   2. Для коммуникаторов.
   3. Для автонавигаторов.
   4. Для томографа.
3) Установка ОС MeeGo в основном производится с помощью этого носителя:
   1. USB Flash.
   2. CD диска.
   3. DVD диска.
   4. Blue - ray диск.
4) Размер стандартного образа MeeGo равен:
   1. 820 Mb.
   2. 10 Mb.
   3. 1 Gb.
   4. 300 Mb.
5) Для того чтобы запустить ОС MeeGo с флэшки необходимо:
   1. Запустить образ в ОС Window.
   2. Перезагрузить компьютер и просто вставить флэшку в USB.
   3. Поставить в BIOS первоприоритетное право загрузки (BOOT) с USB.
   4. MeeGo с флэшки не запустится.
6) Какой слой отсутствует в ОС MeeGo?
   1. Основной.
   2. Средний.
   3. Пользовательский.
   4. Нижний.

Список литературы

1. http://wiki.meego.com

4. Общие средства разработки приложений под Linux

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

4.1. 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.

4.2. 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 (ПО) 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 часто используемые программистами инструменты, разработанные в проекте GNU и являющиеся неотъемлемой частью дистрибутивов операционной системы Linux.

4.2.1. Описание GNU autotools

Как уже было упомянуто выше, GNU autotools состоит из некоторого стандартного набора утилит, которые предназначены для компиляции исходного кода программы под целевую платформу. При помощи них автоматизировано 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 [1].

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.

4.2.2. 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 C. 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 в последствии 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.

4.2.3. 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.

4.2.4. 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.

4.3. 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.

4.4. 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:

- Qt Creator — 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 Qt Designer и Qt Assistant 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.

4.5. 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.

4.6. Лабораторная работа № 2 «Знакомство с традиционными средствами разработки в ОС Linux»

4.6.1. Цель лабораторной работы

Основными целями лабораторной работы являются:

- знакомство с традиционными средствами разработки для Linux

(прежде всего, знакомство с компилятором gcc и системой сборки GNU make)

- сборка программных пакетов из исходного кода
- знакомство с системой управления пакетов на примере дистрибутива Ubuntu.

4.6.2. Введение

Необходимые для выполнения лабораторной работы навыки и программно-аппаратные средства

Для успешного выполнения лабораторной работы читателю желательно иметь базовые навыки программирования на языках C/C++ и быть знакомым с основными служебными программами Linux (ls, rm, mkdir и т. п.).

Перед выполнением необходимо установить ОС Linux. Операционная система может быть установлена на отдельный ПК или запущена в виртуальной машине. При выборе дистибутива ОС лучше отдать предпочтение ОС с развитой системой управления пакетами (dpkg, rpm). В частности, разработка для ОС MeeGo может вестись на ПК под управлением ОС MeeGo. Следует отметить, что указания, представленные в этой лабораторной работе, были проверены на дистрибутиве Ubuntu.

Установка необходимого программного обеспечения

Приложения для управления пакетами отличаются в зависимости от используемого дистрибутива:

1. при использовании системы пакетов dpkg (в дистрибутивах семейства Debian — например, в Ubuntu) можно воспользоваться фронт-эндом apt-get или aptitude.

```
apt-get install <название пакета>
aptitude install <название пакета>
```

Можно также использовать графическое приложение synaptic.

2. при использовании системы управления пакетами rpm

```
yum install <название пакета>
```

Следует заметить, что команды надо исполнять с правами пользователя `root`.

Например,

```
sudo apt-get install make.
```

***Задание*:** Установите пакеты make и gcc.

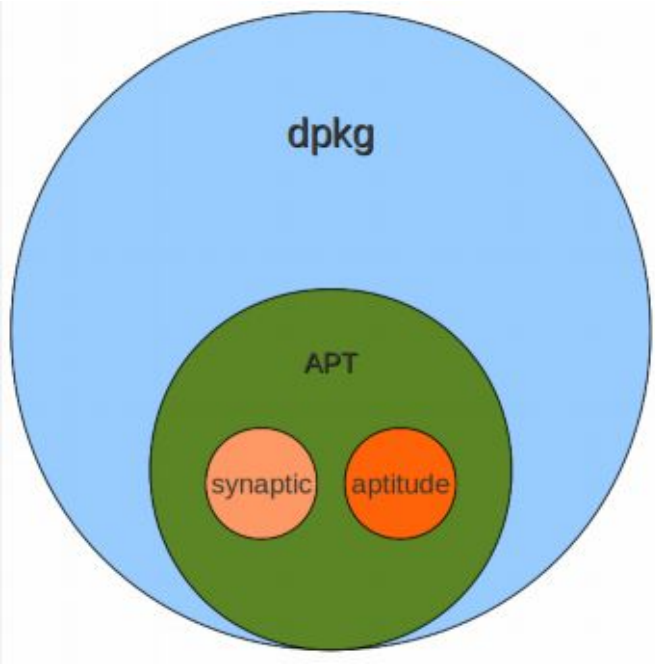


Рис. 4.5.1.

4.6.3. Пакет компиляторов GNU Compiler Collection. Примеры компиляции и линковки простейших приложений

Компиляция простейшего одномодульного приложения

Рассмотрим пример компиляции простейшей программы HelloWorld. Код примера может быть найден в каталоге *lab02/01 файла* labMeeGo.rar.

В простейшем случае для успешного выполнения программы необходимо выполнить следующие шаги:

```
gcc hello.c
```

– по устоявшейся традиции вывод генерируется в файл `a.out`
– запустить полученный исполняемый файл: `./a.out`

Можно задать название выходного файла, отличное от `a.out`.

- Выполните команду

```
gcc -o hello hello.c
```

- Для выполнения скомпилированной программы выполните

```
./hello
```

Компиляция многомодульного приложения

Теперь рассмотрим сборку многомодульного приложения, пример которого находится в каталоге *lab02/02 файла* labMeeGo.rar. В этом случае для получения исполняемого кода необходимо выполнить следующие шаги:

- компиляция и ассемблирование модулей

```
gcc -c -o hello_main.o hello_main.c
gcc -c -o hello_util.o hello_util.c
```

- линковка модулей в один исполняемый файл

```
gcc -o hello hello_main.o hello_util.o
```

Используемый нами на первом этапе ключ -c сообщает gcc, что не следует выполнять линковку компилируемого модуля. Это позволяет компилировать модули, имеющие зависимости от других модулей, которые будут разрешены на втором этапе.

Линковка с внешними библиотеками

В каталоге *lab02/03 файла* labMeeGo.rar. находится пример кода, имеющего зависимости от внешних библиотек. В процессе его сборки потребуется линковка с этими библиотеками. Для компиляции этого кода необходимо:

- установить библиотеку ncurses

```
apt-get install libncurses-dev
```

- компилировать и ассемблировать модули (аналогично предыдущему примеру)
- прилинковать модули и библиотеку ncurses в один исполняемый файл

```
gcc -lncurses -o hello hello_main.o hello_util.o
```

По умолчанию компилятор gcc выполняет динамическую линковку: в созданном исполняемом файле указывается ссылка на необходимую библиотеку и перечень символов (функций, переменных) этой библиотеки, используемых в компилируемом коде. При вызове полученного исполняемого файла загрузчик ОС прочитает эти зависимости и попытается найти в одном из заданных заранее каталогах (например, `/usr/lib`) соответствующую библиотеку динамической линковки. Обычно это файл с расширением `*.so` (от «shared object») и названием, совпадающим с названием библиотеки, указанным при помощи ключа `-l`, за исключением префикса «`lib`». К примеру, в нашем случае будет искаться файл `libncurses.so`. Обнаружив необходимую библиотеку, загрузчик поместит её в память, а также заместит в загруженном в память образе исполняемого файла ссылки на внешние функции указателями на конкретные адреса в памяти, в соответствии с адресом загрузки библиотеки. В случае, если необходимая библиотека не будет обнаружена, будет выдана ошибка загрузчика и программа не будет выполнена.

При необходимости, вместо динамической линковки может быть выполнена статическая. В этом случае линковщик попытается найти библиотеку статической линковки. Обычно это файл с расширением `*.a` и названием, сформированным по тому же принципу, что и название файла библиотеки динамической линковки. Если необходимый файл найден, линковщик поместит целиком его исполняемый код в выходной исполняемый файл, подобно тому, как в него помещаются модули компилируемого проекта.

- Для сборки скомпилированных на предыдущем этапе объектов со статической линковкой с библиоткой ncurses, выполните

```
gcc -o hello_static hello_main.o hello_util.o -static -lnucrses
```

Применение статической линковки в большинстве случаев неоправданно, так как размер исполняемого файла при этом сильно увеличивается. Сравните размеры полученных в этом примере исполняемых файлов `hello` и `hello_static`, вызвав команду

```
ls -lh hello hello_static
```

Компиляция кода на C++

Пакет компиляторов gcc способен компилировать код для C, C++, Objective C, Fortran и т. д. Используемый язык программирования определяется по расширению файла, но может также быть задан при помощи ключа компилятора. Поэтому в примере

```
gcc -c myfile.cpp
```

где файл `myfile.cpp` — это код на C++, компиляция пройдет успешно. Однако, если мы опустим ключ `-c`, сборка завершится с многочисленными ошибками линковки. Это вызвано тем, что при сборке при помощи gcc, автоматически выполняется линковка со стандартной библиотекой C, но не со стандартной библиотекой C++. Недостающая зависимость может быть указана при помощи ключа компилятора, однако для удобства обычно используется `g++` — обёртка вокруг компилятора `gcc`. При её использовании связь со стандартной библотекой C++ указывается автоматически. Синтаксис `g++` идентичен синтаксису `gcc`.

Наиболее часто используемые ключи gcc

Для управления процессом предобработки кода, его компиляции, ассемблирования и линковки могут использоваться ключи командной строки. Общее число ключей, понимаемых gcc, чрезвычайно велико; узнать о них можно из руководство (man-страницы) по gcc и из полной документации. Мы перечислим лишь некоторые из часто используемых ключей.

- При помощи ключа `-g` в исполняемый файл может быть добавлена отладочная информация.
  - В качестве примера соберем пример из каталога *lab02/01 файла* labMeeGo.rar:

```
gcc -g -o hello hello.c.
```

  - Теперь изучим ассемблерный листинг полученного исполняемого файла с привязкой к строкам кода:

```
objdump -S hello | less
```

- Вызов компилятора `gcc` с ключом `-Wall` приведет к включению всех предупреждений, т. е. компилируемый код будет проверяться по расширенному набору правил. Для примера из каталога *lab02/01 файла* labMeeGo.rar сборка с включением всех предупреждений будет выглядеть следующим образом:

```
gcc -Wall -o hello hello.c
```

- При помощи ключа `-I` можно указать каталоги, в которых будут искаться заголовочные файлы.
- Ключ `-L` позволяет задать каталоги, в которых линковщик будет искать библиотеки.
- Ключ `-D` позволяет определить макроконстанту. В качестве примера определим макроконстанту DEBUG:

```
gcc -DDEBUG -o hello hello.c.
```

После ее определения в коде можно использовать определить, например, блок условной компиляции при помощи директивы макропроцессора `#ifdef`:

```
#ifdef DEBUG
… some code
#endif
```

В этом случае код, помещённый между директивами `#ifdef` и `#endif` будет скомпилирован лишь в том случае, если при компиляции будет определена макроконстанта `DEBUG`.

- gcc использует набор правил, позволяющих оптимизировать компилируемый код по ряду параметров — таких, как время исполнения, используемая память, размер исполняемого файла. Любое правило оптимизации может быть включено и отключено при помощи отдельного ключа, но для удобства часто используются 4 «уровня» оптимизации, каждому из которых соответствует определённый набор правил оптимизации.
  - Задание уровня оптимизация кода выполняется при помощи ключей `-O0`, `-O1`, `-O2`, `-O3`, которые активизируют различные уровни оптимизации. Чем выше уровень, тем больше множество включенных правил оптимизации.
- Ключ `-Os` отвечает за минимизацию размера итогового двоичного файла
- При том, что конкретный экземпляр gcc компилирует в двоичный код для определенного семейства процессоров, он может задействовать правила оптимизации, использующие особенности конкретной архитектуры процессора из этого семейства. Ключи `-mtune=cpu`, `-march=cpu`, где `cpu` — имя конкретной архитектуры, позволяют задействовать правила оптимизации для определённой архитектуры. Так, например, при компиляции с ключом `-mtune=core2` будут включены правила оптимизации, использующие особенности процессоров линейки Core2

4.6.4. Система сборки GNU make

4.6.4.1. Запуск сборки с использованием GNU make

Задание находится в каталоге lab02/04/sample файла labMeeGo.rar.

Для сборки проекта при помощи GNU make необходимо выполнить следующие шаги:

- выполните команду make
  - make ищет файлы с названием makefile, Makefile...
  - будет собрана первая по порядку описания в файле цель (target)
- выполните команду

```
make -f Makefile.3
```

в качестве makefile будет использоваться файл Makefile.3

- выполните команду

```
make -f Makefile.3 clean
```

будет собрана цель clean

- выполните команду

```
make -f Makefile.3 -j 2
```

Сборка будет выполняться параллельно, т. е. make будет запускать одновременно по две команды.

4.6.4.2. Создание простого Makefile

Задание находится в каталоге lab02/04/task файла labMeeGo.rar*:*

***Задание*:** По примерам make-файлов в подкаталоге sample написать make-файлы для проекта “ncurses Snake” в подкаталоге task.

Этап 0

Руководствуясь примером «Пример Makefile.0» в подкаталоге lab02/04/task файла labMeeGo.rar*, создайте makefile для проекта в подкаталоге task*

- Формат правила:

```
target: prerequisite1 ... prerequisiteN
recipe
…
```

  - *target* – какую цель (какой файл) мы хотим получить на данном этапе
  - *prerequisites* – какие цели (какие другие файлы) нам для этого требуются
  - *recipe* – какие команды надо выполнить, чтоб получить target

Этап 1

Руководствуясь примером «Пример Makefile.1» в подкаталоге lab02/04/task файла labMeeGo.rar*, создайте makefile для проекта в подкаталоге task*

- Добавляем цели для сборки всего проекта и для удаления результатов предыдущей сборки, традиционно называющиеся all и clean. Цели all и clean объявляются как .PHONY: им не соответствуют реальные файлы. Это избавит нас от неприятностей, если в директории сборки окажутся файлы с такими названиями.

- В командах используем особые переменные:
  @ – разворачивается в название цели
  < – разворачивается в первый элемент списка реквизитов
  ^ - разворачивается в список реквизитов.

Этап 2

Руководствуясь примером «Пример Makefile.2» в подкаталоге lab02/04/task файла labMeeGo.rar*, создайте makefile для проекта в подкаталоге task*

- Параметризуем команды сборки при помощи переменных. Запустите:

```
make -f Makefile.2 CFLAGS='-O0 -g'
```

- Убедитесь, что компиляция будет запускаться именно с указанными ключами -O0 и -g.

Этап 3

Руководствуясь примером «Пример Makefile.3» в подкаталоге lab02/04/task файла labMeeGo.rar, *создайте makefile для проекта в подкаталоге task*

- Используем implicit rules:
  Система сборки make позволяет опускать некоторые типовые правила, которые будут формироваться автоматически. В этом случае правило будет определено на основании расширения файла цели. Например:
  - recipe для цели `n.o`, если существует файл `n.c`, формируется как:

```
$(CC) $(CPPFLAGS) $(CFLAGS) -c -o n.o n.c
```

  - recipe для цели n.o, если существует один из файлов `n.cc`, `n.cpp` или `n.c`:

```
$(CXX) $(CPPFLAGS) $(CXXFLAGS) -c -o n.o n.cpp
```

4.6.5. Сборка программных пакетов из исходного кода

Сборка и установка открытых программных программных пакетов для linux в большинстве случаев чрезвычайно проста. Убедимся в этом на примере пакета bash — широко распространенного интерпретатора командной строки, рабрабатываемого в рамках проекта GNU.

- Скачиваем с сайта GNU исходные коды последней версии bash:

```
wget http://ftp.gnu.org/gnu/bash/bash-4.1.tar.gz
```

- Распаковываем архив приложением tar:

```
tar xzf bash-4.1.tar.gz
```

- В полученном каталоге запускаем конфигурационный скрипт:

```
./configure
```

можно запустить `configure --help` для того, чтобы узнать, какие аргументы можно передать команде configure

- Запускаем сборку командой `make all`
- *Опционально:* устанавливаем пакет командой `make install`

4.6.6. Добро пожаловать в мир открытого кода

Большинство современных систем управления пакетами предусматривают работу как с репозиториями установочных пакетов, так и с репозиториями пактов исходного кода. Благодаря этому установка программных пакетов из исходного кода становится столь же простой как и установка этих пакетов из установочных архивов. Установить практически любой пакет из доступных в репозиториях для вашего дистрибутива linux можно следующим образом:

- Скачиваем исходники при помощи менеджера пакетов:

```
apt-get source <название пакета>
```

- Устанавливаем зависимости — пакеты, необходимые для сборки вашего пакета:

```
sudo apt-get build-dep <название пакета>
```

- Конфигурируем, собираем и устанавливаем пакет.

Дополнительная информация

- man pages (руководство): просмотр вызывается командой

```
man PAGE
```

  где PAGE – название страницы руководства. Например:

```
man gcc
```

4.6.7. Задания для самостоятельной работы

1. Скачайте и соберите пакет grep — инструмент для поиска в текстовых файлах по регулярным выражениям.

4.7. 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.

4.8. 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) Где и когда впервые была разработана утилита make?
1. Bell Labs, 1977
2. проект GNU, 1985
3. корпорация Apple, 1995
4. Intel, 1990

23) 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

24) 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

25) 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

26) 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

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

28) 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

29) 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

30) 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

31) 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.

32) 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

33) 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

34) 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>

35) 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

36) Что из перечисленного не является системой автоматизации сборки?
1. qmake
2. Qt
3. Cmake
4. Scons

37) Как расшифровывается аббревиатура gdb?
1. Greedy Disk-Conserving Broadcasting
2. Genome Database

3. GNU database
4. GNU debugger

38) Кем и когда была выпущена первая версия gdb?
1. Ричардом Столлманом в 1986
2. Джоном Джилмором в 1992
3. Марком Шатлвортом в 2001

39) Отладку какого из перечисленных языков не поддерживает отладчик gdb?
1. C
2. Fortran
3. Modula-2
4. Haskell

40) Какой из перечисленных ниже инструментов предназначен для обнаружения утечек памяти в коде?
1. KcacheGrind
2. perfmon
3. Valgrind
4. strace

41) Какой инструмент требует перекомпиляции ядра для своей работы?
1. KcacheGrind
2. perfmon
3. Oprofile
4. PTLsim

42) 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.

5. MeeGo SDK. Обзор технологии Qt

Nokia Qt SDK – инструмент для кросс-платформенной разработки. Интеловские инструменты для разработки программного обеспечения для мобильных устройств и элементы технологии разработки. Сообщества разработчиков под ОС MeeGo и для процессора Интел Atom. Примеры разработки приложения “Hello Word”.

5.1. 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.

5.2. 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;
- MeeGo под QEMU;
- MeeGo в chroot среде;
- Кросс-компиляция и удаленная отладка на целевом устройстве.

Опишем вкратце каждый из них.

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. Разрабатываемое 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.

5.3. 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.

5.3.1. 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).

5.3.2. 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.

5.3.3. 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).

5.3.4. Инструменты разработки на Qt

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

- Qt Creator – 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 Qt Designer и Qt Assistant и графический фронтенд для gdb. Qt Creator входит в состав SDK.
- QtDesigner – инструмент для визуального дизайна графических интерфейсов. В результате работы QtDesigner создается xml файл, описывающий графический интерфейс.
- QtLinguist – локализация интерфейса.
- QtAssistant – система справки.

- Qt Simulator – 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.

5.3.5. 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.

5.3.6. 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. 5.3.1).

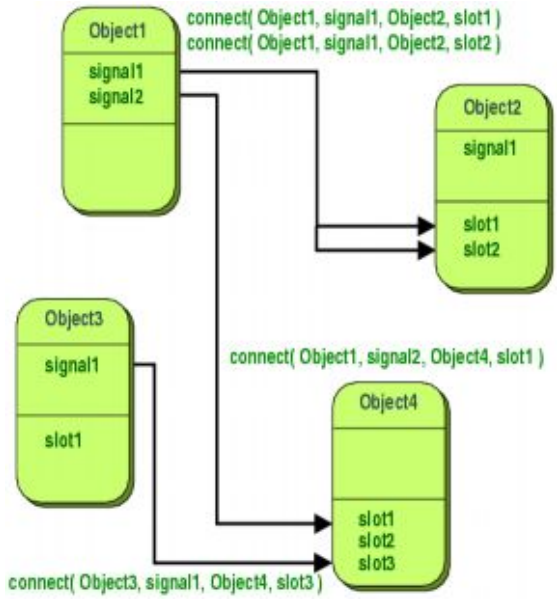


Рис. 5.3.1.

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.

5.4. Лабораторная работа № 3 «Знакомство с MeeGo SDK»

5.4.1. Цель лабораторной работы

Научиться устанавливать на компьютер и пользоваться MeeGo SDK.

5.4.2. Введение

План работы

- Установка MeeGo SDK
- Работа с MeeGo SDK
  - Создание приложения в qtcreator
  - Его сборка и запуск.

Необходимые навыки

- Базовое знание языков программирования C/C++.

- Желательно базовое знакомство с основными служебными программами Linux (ls, rm, mkdir и т. п.)

Необходимые программные и аппаратные средства

- ПК под ОС Linux (поддерживаются дистрибутивы Fedora 13, Ubuntu 10.04, openSUSE 11.3).
- Желательно наличие GPU от Intel для эмуляции графического интерфейса MeeGo.

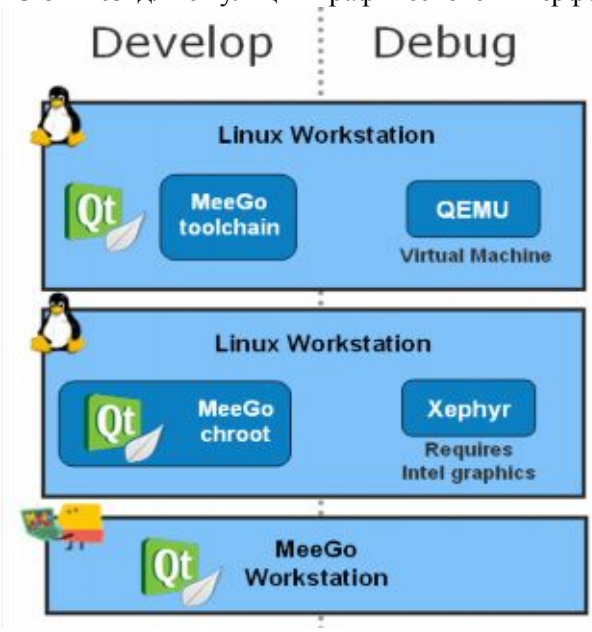


Рис. 5.4.1.

5.4.3. Установка MeeGo SDK

MeeGo SDK – варианты развёртки

- Разработка ведётся сразу на целевом устройстве под MeeGo.
- Виртуализация: MeeGo исполняется на эмуляторе QEMU.
- chroot: MeeGo исполняется в нативной среде, но в изолированной песочнице.
- Разработка и компиляция на любой машине под Linux. Удалённая отладка на целевом устройстве.

Развёртка MeeGo SDK в варианте с chroot

chroot – стандартное служебное приложение в Unix-подобных системах, позволяющее запускать программы с заданием произвольного каталога файловой системы в качестве корневого каталога. Запущенная таким образом программа будет видеть только те файлы и подкаталоги исходной файловой системы, которые были помещены в указанный каталог. Этот инструмент активно используется для создания «песочниц» внутри основной файловой системы.

Фактически, MeeGo SDK версии 1.0 представляет собой образ файловой системы с установленным MeeGo, а также инструментами для разработки, необходимыми библиотеками и заголовочными файлами.

Таким образом, выполнение, chroot в каталог, к которому был подключён (mounted) образ MeeGo SDK, позволяет работать в «песочнице» с установленным MeeGo.

Скачиваем MeeGo SDK 1.0 и chroot-скрипт

Скачайте образ MeeGo SDK 1.0 с сайта MeeGo в одной из двух версий:

- для нетбука:
  - http://download3.meego.com/sep09/meego-netbook-ia32-1.0.80.12.20100727.1-sdk-pre0901.raw.tar.bz2

- для PDA (handset)
  - http://download3.meego.com/meego-handset-sdk-20101012-1.1.80.20101024.1603-sda.raw.tar.bz2

  Скачайте скрипт для выполнения chroot:
  - http://download3.meego.com/meego-sdk-chroot

  Актуальные ссылки для скачивания доступны на странице wiki:
  - http://wiki.meego.com/Getting_started_with_the_MeeGo_SDK_for_Linux

Распаковка образа и его подключение (mounting)

- Распакуйте образ SDK при помощи tar

```
tar xvjf  <имя файла с архивом>
```

Внимание: команда будет выполняться продолжительное время.

- Создайте каталог, к которому будет подключен образ

```
mkdir  <имя каталога>
```

- Подключите (mount) образ в созданную директорию

```
sudo mount -o loop,offset=512 <путь к файлу образа> <путь к директории>
```

Выполнение chroot

- Сделайте скрипт meego-sdk-chroot исполняемым при помощи служебного приложения chmod:

```
chmod +x <путь к скрипту>
```

- Выполните скрипт chroot:

```
sudo ./meego-sdk-chroot <путь к директории>
```

- Обратите внимание на приглашение для ввода: вновь запущенный интерпретатор командной строки видит в качестве корня ОС каталог с SDK. Это также означает, что любые приложения, запущенные из этого экземпляра интерпретатора, будут исполняться внутри песочницы MeeGo SDK с её набором библиотек и приложений.

Эмуляция графической оболочки MeeGo

Стандартным методом для построения графических пользовательских интерфейсов во многих *nix-системах является де-факто оконная система X Windows. Эта система отделяет графическое приложение от подсистемы, отвечающей за отрисовку интерфейса и за его представление пользователю, используя при этом клиент-серверную модель. X-сервер — это серверная часть оконной систем, представляющая некое виртуальное устройство, способное представлять пользователю набор из графических примитивов. Для эмуляции графической оболочки MeeGo при развёртывании SDK под chroot применяется Xephyr – X сервер, использующий для вывода окно другого X сервера. Проще говоря, Xephyr создаёт виртуальный дисплей в окне, представленном на другом дисплее.

Внимание: для работы с Xephyr необходим GPU от Intel.

- Запустите графическую оболочку MeeGo, выполнив в среде chroot:

```
startmeego &
```

- Для того, чтоб запустить произвольное приложение в среде chroot так, чтоб оно использовало для отображения GUI окно Xephyr, необходимо установить переменную окружения DISPLAY

```
export DISPLAY=:2
```

- При значении переменной DISPLAY равном :0, запущенное приложение будет использовать основной дисплей (но при этом – графические библиотеки MeeGo)

  Задание: запустите текстовый редактор gedit на основном дисплее и на дисплее Xephyr

Важное предупреждение – MeeGo SDK 1.1

Хотя варианты развёртки MeeGo SDK в версии 1.1 остались прежними, был полностью пересмотрен способ его установки. Для установки SDK используются репозитории (поддерживаются системы управления пакетами apt, yum, Zypp). Возможна покомпонентная установка SDK. SDK как образа файловой системы с MeeGo в новой версии не существует. Для разработки с использованием

chroot **в настоящий момент** предлагается либо использовать образы SDK версии 1.0, либо самостоятельно построить среду chroot.

5.4.4. Примеры работы с Qt. Создание простейших приложений в qtcreator

Сборка и запуск консольного приложения

Пример находится в каталоге lab03/01 файла labMeeGo.rar.

- Скопируйте пример в chroot-среду
- откройте новую вкладку / окно терминала
- скопируйте каталог с примером в каталог, куда был подключён образ SDK:

```
sudo cp -r <путь к каталогу с примером> <путь к каталогу с SDK>/root
```

- В терминале, в котором был выполнен chroot, перейдите в скопированный каталог

```
cd /root/01
```

- Сгенерируйте файл проекта при помощи qmake:

```
qmake -project
```

- Изучите созданный файл .pro (он будет называться 01.pro по имени каталога)
- Сгенерируйте makefile из файла .pro

```
qmake
```

Вызов qmake без аргументов эквивалентен вызову qmake -makefile

- Убедитесь, что был создан Makefile. Изучите его содержимое на основании материала лабораторной работы №2.
- Запустите сборку

```
make (или make all)
```

- Убедитесь, что был создан исполняемый файл (т. к. название цели сборки не было указано нами в проектном файле, он будет называться 01 по имени файла .pro)

Генерация заглушки

Следующие команды следует выполнять в среде chroot.

- Установите значение переменной DISPLAY в ':0' для того, чтоб для для отображения GUI qtcreator не использовался Xephyr (см. 5.2)
- Запустите qtcreator

```
qtcreator &
```

- Создайте новое консольное приложение:
- Выберите в меню File->New File or Project
- Выберите вариант Qt Application Project->Qt Gui Application
- Выберите название проекта и путь к каталогу, в котором он будет создан
- Примите значения по умолчанию для данных о классах
- Нажмите на Finish для завершения

Работа с редактором форм

- Войдите в режим редактора, выбрав Edit в навигационной панели слева
- Откройте для редактирования файл формы mainwindow.ui
- Перетащите на панель виджета элементы управления Label и Push Button из списка виджетов слева
- Двойным щелчком по Label войдите в режим редактирования текста. Измените текст на любой по вашему усмотрению.
- При помощи редактора свойств в правом нижнем углу измените свойство Font элемента label и увеличьте шрифт ярлыка.

Работа с сигналами и слотами

Для дополнительной информации см. текст лекции №5.

- В режиме «Edit» откройте для редактирования заголовочный файл mainwindow.h
- В классе MainWindow объявите слот `toggleLabelVisibility()`
- В файле mainwindow.cpp реализуйте toggleLabelVisibility так, чтобы при вызове он изменял состояние видимости элемента label
  - состояние видимости label можно получить посредством вызова метода

```
ui->label->isVisible()
```

  - состояние видимости label можно установить вызовом метода

```
ui->label->setVisible(bool)
```

- В конструкторе MainWindow подключите сигнал `clicked()` элемента pushButton к слоту `toggleLabelVisibility()`

```
connect(ui->pushButton, SIGNAL(clicked()), this, SLOT(toggleLabelVisibility())
```

5.4.4.1. Настройка и запуск

- Для того, чтоб созданное нами приложение запускалось в графической оболочке MeeGo в окне Xephyr (напомним, это возможно, если ваша система использует GPU от Intel) следует:
  - запустить графическую оболочку MeeGo

```
startmeego &
```

  - в qtcreator перейти в режим Projects (панель слева)
  - на вкладке «Run Settings» развернуть раздел «Run Environment»
  - установить значение переменной DISPLAY в ':2'
- Сохраните все файлы проекта нажатием Ctrl+Shift+S
- Соберите и запустите приложение нажатием на зелёную стрелку в левом нижнем углу окна
- Убедитесь, что нажатие на кнопку меняет состояние видимости ярлыка.

5.5. 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.

5.6. 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

Qt Documentation
3. http://doc.qt.nokia.com/4.7/index.html.

6. ОС MeeGo “изнутри”

Ядро ОС MeeGo и состав компонент. MeeGo 1.0 release. Состав и функции ядра (Setting Database [GConf], System Libraries [glibc, glib, …], Message Bus [D-Bus], Platform Info [libudev]). Сборка, загрузка, инсталляция.

6.1. Введение

Эта лекция посвящена начальному знакомству с архитектурой операционных систем семейства Linux, в частности, с архитектурой MeeGo. Архитектура современной операционной системы чаще всего устроена так, что разработчикам пользовательских приложений не приходится использовать знания о ней. Но в то же время для выполнения различных тонких настроек кода, его оптимизации такие знания нередко оказываются ключевыми.

Сегодня мы кратко рассмотрим архитектуру ядра ОС Linux, историю создания его основных компонент и подсистем; обсудим важные подсистемы ОС MeeGo, такие как D-Bus и графическую оболочку.

6.2. Ядро операционной системы Linux

Проект Linux занимается разработкой ядра операционной системы, которое используется совместно с инструментами, разработанными в рамках проекта GNU, образуя, таким образом, полную ОС, знакомую нам как ОС GNU/Linux. Ядро операционной системы представляет из себя центральную часть ОС, абстракцию над аппаратурой системы, выполняется в привилегированном режиме и в отдельном адресном пространстве. Зачастую определение ядра как программы, выполняющейся в привелигированном режиме является верным. Пользовательские приложения при этом выполняются в другом адресном пространстве и контролируются ядром. Основные компоненты ядра Linux – это системы управления памятью, управления аппаратной частью (управление драйверами устройств), управления процессами, файловой системой, диспетчер процессов, позволяющий справедливо разделять процессорное время между процессами.

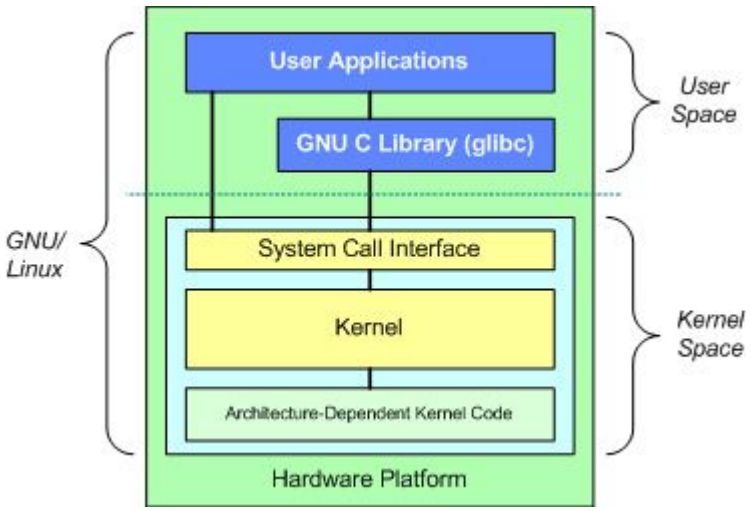


Рис. 6.2.1.

В Linux взаимодействие пользовательских приложений и ядра происходит по интерфейсу SCI (System call interface), при этом в большинстве случаев пользовательские приложения напрямую к ядру не обращаются, так как интерфейс взаимодействия с ядром достаточно сложен. Для

взаимодействия с ядром приложения используют, главным образом, интерфейс стандартной библиотеки glibc (GNU C Library). С его использованием большинство операций, требующих обращения к ядру ОС становятся значительно проще. Так, например, выделение памяти может быть выполнено при помощи простого вызова функции `malloc`, открытие файла — при помощи вызова `fopen`.

6.2.1. История создания ядра ОС Linux

Разработка операционной системы Linux была начата студентом Хельсинского университета Линусом Торвальдсом в апреле 1991. Прототипом для будущего ядра стала операционная система MINIX: совместимая с UNIX операционная система для персональных компьютеров, которая загружалась с дискет и умещалась в очень ограниченной в те времена памяти персонального компьютера. MINIX был создан Эндрю Таненбаумом в качестве учебной операционной системы, демонстрирующей архитектуру и возможности UNIX, но непригодной для полноценной работы.

Ядро Linux разрабатывалось на C, изначально — для популярной архитектуры Intel 80386. Первый релиз Linux был сделан в сентября 1991 года. Программа состояла всего из 10239 строк кода. Разработка ядра Linux началась как просто хобби одного человека, но вскоре стало ясно, что необходимость в ядре, интергрированным с инструментами проекта GNU, велика. И дальнейшее развитие операционной системы проходит весьма бурно.

Текущая версия Linux – версия 2.6.36, которая насчитывает уже более 13 миллионов строк кода и поддерживает свыше 20 архитектур. Начиная с версии 0.99 ядро распространяется под лицензией GPL.

6.2.2. Процесс разработки ядра

Несмотря на то что только 2% кода операционной системы Linux написано Линусом Торвальдсом, его участие в разработке нельзя назвать условным. Он полностью контролирует процесс разработки и принимает все стратегические решения. Он же одобряет патчи, которые войдут в текущий релиз.

Для разработки ОС Linux используется система версионного контроля git. Основным интерфейсом общения разработчиков является почтовая рассылка (http://www.kernel.org/pub/linux/docs/lkml/). Сюда выкладываются все предлагаемые изменения и написанные патчи, которые рецензируются руководителями, а затем, возможно, включаются в следующий релиз ядра. Необходимо заметить, что значительная часть патчей носит сугубо экспериментальный характер и, быть может, никогда не будут включены в основную ветку.

6.2.3. Строение ядра Linux

Основные структурные компоненты ядра Linux отображены на Рис.6.2.2.

Интерфейс системных вызовов

SCI – это тонкий уровень, предоставляющий средства для вызова функций ядра из пространства пользователя. Этот интерфейс может быть архитектурно зависимым, даже в пределах одного процессорного семейства. SCI фактически представляет собой службу мультиплексирования и демультиплексирования вызова функций. Реализация SCI находится в ./linux/kernel, а архитектурно-зависимая часть - в ./linux/arch.

Управление процессами

Как показано на Рис. 6.2.2 ядро по сути представляет собой диспетчер ресурсов. Независимо оттого, что представляет собой управляемый ресурс - процесс, память или аппаратное устройство, - ядро организует и упорядочивает доступ к ресурсу множества конкурирующих пользователей (как в пространстве ядра, так и в пространстве пользователя).

Управление процессами сконцентрировано на исполнении процессов. В ядре эти процессы называются потоками (threads); они соответствуют отдельным виртуализованным объектам

процессора (код потока, данные, стек, процессорные регистры). В пространстве пользователя обычно используется термин процесс, хотя в реализации Linux эти две концепции (процессы и потоки) не различают. Ядро предоставляет API для создания нового процесса (порождения копии, запуска на исполнение, вызова функций Portable Operating System Interface [POSIX]), остановки процесса (kill, exit), взаимодействия и синхронизации между процессами (сигналы или механизмы POSIX).

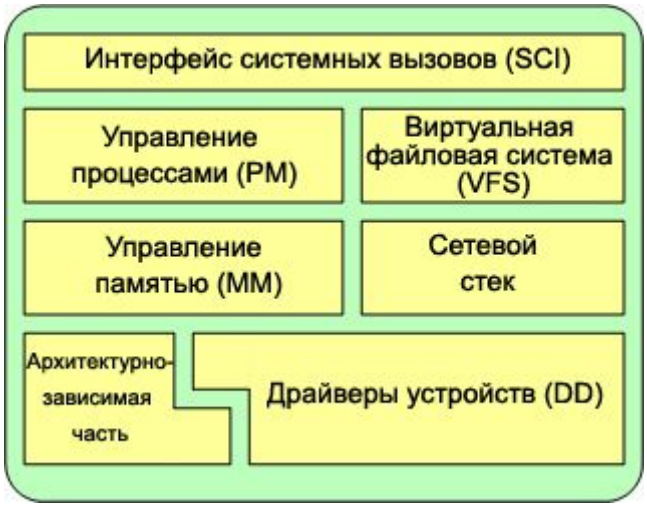


Рис.6.2.2.

Еще одна задача управления процессами – совместное использование процессора активными потоками. В ядре реализован новаторский алгоритм планировщика, время работы которого не зависит от числа потоков, претендующих на ресурсы процессора. Название этого планировщика - O(1) - подчеркивает, что на диспетчеризацию одного потока затрачивается столько же времени, как и на множество потоков. Планировщик O(1) также поддерживает симметричные многопроцессорные конфигурации (SMP). Исходные коды системы управления процессами находятся в ./linux/kernel, а коды архитектурно-зависимой части - в ./linux/arch).

Управление памятью

Другой важный ресурс, которым управляет ядро - это память. Для повышения эффективности, учитывая механизм работы аппаратных средств с виртуальной памятью, память организуется в виде т.н. страниц (в большинстве архитектур размером 4 КБ). В Linux имеются средства для управления имеющейся памятью, а также аппаратными механизмами для установления соответствия между физической и виртуальной памятью.

Однако управление памятью - это значительно больше, чем просто управление буферами по 4 КБ. Linux предоставляет абстракции над этими 4 КБ буферами, например, механизм распределения slab allocator. Этот механизм управления базируется на 4 КБ буферах, но затем размещает структуры внутри них, следя за тем, какие страницы полны, какие частично заполнены и какие пусты. Это позволяет динамически расширять и сокращать схему в зависимости от потребностей вышележащей системы.

В условиях наличия большого числа пользователей памяти возможны ситуации, когда вся имеющаяся память будет исчерпана. В связи с этим страницы можно удалять из памяти и переносить на диск. Этот процесс обмена страниц между оперативной памятью и жестким диском называется подкачкой. Исходные коды подсистемы управления памятью находятся в ./linux/mm.

Виртуальная файловая система

Еще один интересный аспект ядра Linux – виртуальная файловая система (VFS), которая предоставляет общую абстракцию интерфейса к файловым системам. VFS предоставляет уровень коммутации между SCI и файловыми системами, поддерживаемыми ядром (Рис. 6.2.3).

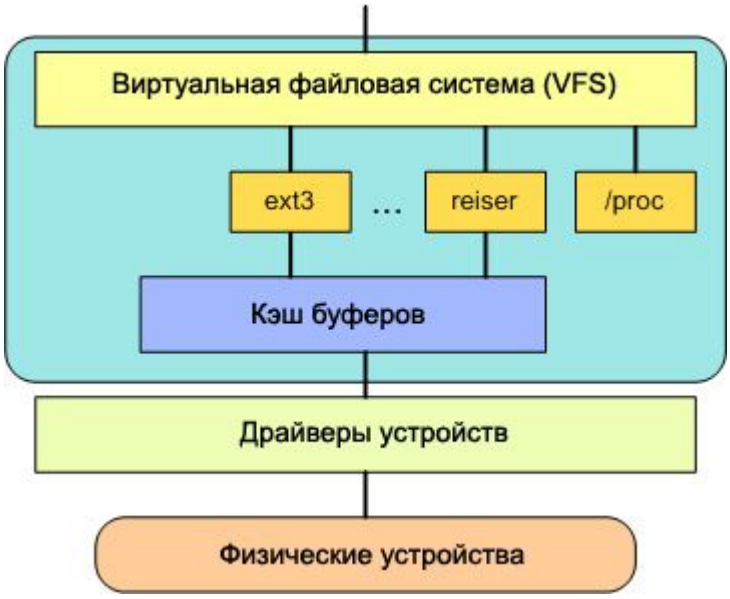


Рис. 6.2.3. VSF.

На верхнем уровне VFS располагается единая API-абстракция таких функций, как открытие, закрытие, чтение и запись файлов. На нижнем уровне VFS находятся абстракции файловых систем, которые определяют, как реализуются функции верхнего уровня. Они представляют собой подключаемые модули для конкретных файловых систем (которых существует более 50). Исходные коды файловых систем находятся в ./linux/fs.

Ниже уровня файловой системы находится кэш буферов, предоставляющий общий набор функций к уровню файловой системы (независимый от конкретной файловой системы). Этот уровень кэширования оптимизирует доступ к физическим устройствам за счет краткосрочного хранения данных (или упреждающего чтения, обеспечивающего готовность данных к тому моменту, когда они понадобятся). Ниже кэша буферов находятся драйверы устройств, реализующие интерфейсы для конкретных физических устройств.

Сетевой стек

Сетевой стек по своей конструкции имеет многоуровневую архитектуру, повторяющую структуру самих протоколов. Протокол Internet Protocol (IP) – это базовый протокол сетевого уровня, располагающийся ниже транспортного протокола Transmission Control Protocol, TCP). В ядре Linux выше реалиации протокола TCP находится уровень сокетов, вызываемый через SCI.

Уровень сокетов представляет собой стандартный API к сетевой подсистеме. Он предоставляет интерфейс к различным сетевым протоколам. Уровень сокетов реализует стандартизованный способ управления соединениями и передачи данных между конечными точками, от доступа к "чистым" кадрам данных и блокам данных протокола IP (PDU) и до протоколов TCP и User Datagram Protocol (UDP). Исходные коды сетевой подсистемы ядра находятся в каталоге ./linux/net.

Драйверы устройств

Подавляющее большинство исходного кода ядра Linux приходится на драйверы устройств, обеспечивающие возможность работы с конкретными аппаратными устройствами. В дереве исходных кодов Linux имеется подкаталог драйверов, в котором, в свою очередь, имеются подкаталоги для различных типов поддерживаемых устройств, таких как Bluetooth, I2C, последовательные порты и т. д. Исходные коды драйверов устройств находятся в ./linux/drivers.

Архитектурно-зависимый код

Хотя основная часть Linux независима от архитектуры, на которой работает операционная система, в некоторых элементах для обеспечения нормальной работы и повышения эффективности необходимо учитывать архитектуру. В подкаталоге ./linux/arch находится архитектурно-зависимая часть исходного кода ядра, разделенная на ряд подкаталогов, соответствующих конкретным архитектурам. Подкаталог для каждой архитектуры содержит ряд вложенных подкаталогов, относящихся к конкретным аспектам ядра, таким как загрузка, ядро, управление памятью и т. д. Исходные коды архитектурно-зависимой части находятся в ./linux/arch.

6.2.4. Сборка ядра

Замечательным свойством Linux-подобных операционных систем является то, что при необходимости можно не только изменить некоторые системные параметры, но и внести изменения в код ядра ОС и перекомпилировать его. Компиляция ядра почти всегда выполняется компилятором gcc. Так как при компиляции используются особые возможности gcc, использование другого компилятора затруднено. В качестве системы управления сборкой используется уже известный нам из предыдущих лекций GNU `make`.

Для успешной компиляции и сборки ядра необходимо пройти следующие этапы:

1. Скачивание исходного кода ядра

Исходный код последней версии ядра для Linux можно найти, например, на ftp://ftp.kernel.org или на любом другом зеркале. Ядра лежат в каталоге /pub/linux/. Необходимо загрузить ядро и поместить его в каталог /usr/src. Там же надо создать каталог для файлов и каталогов ядра (обычно его название соответствует версии ядра, например: linux-2.X.X, где 2.X.X – версия нового ядра) командой mkdir. После этого создается связь с каталогом linux (ln -s linux-2.X.X linux). Затем при помощи программы tar в созданный каталог распаковывается архив с исходным кодом ядра (например: linux-2.X.X.tar.gz).

2. Конфигурирование ядра

На этом этапе выбираются опции, которые будут использоваться в новом ядре. Не обязательно вникать в подробности массы опций. В большинстве из них можно воспользоваться настройками по умолчанию.

Конфигуратор ядра вызывается из корневой директории с исходными кодами ядра при помощи `make`.

`make config` вызывает режим консольного конфигуратора

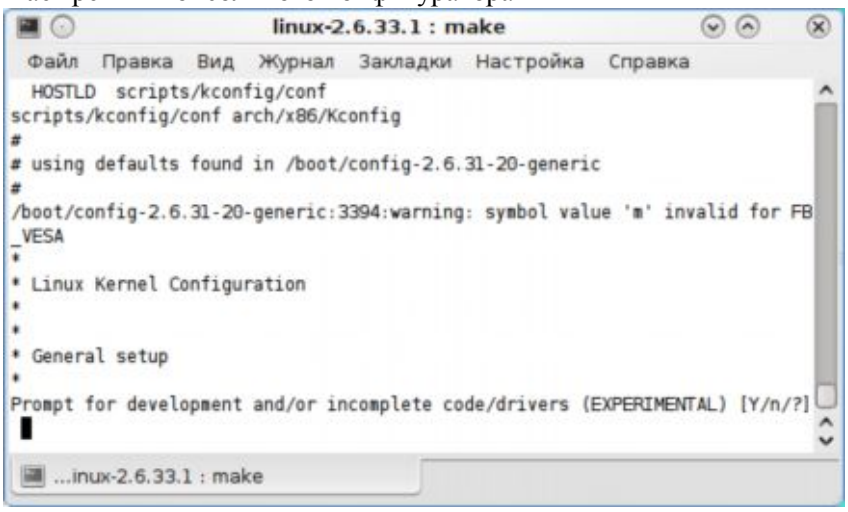


Рис. 6.2.4. `make menuconfig` – консольный режим в виде списка.

После внесения нужных изменений, сохраните настройки и выйдете из конфигуратора.

3. Компиляция ядра

Компиляция и установка ядра выполняется при помощи команд: `make && make install`.

Первая команда скомпилирует в машинный код все файлы, что займет достаточно продолжительное время, а вторая установит новое ядро в систему. После того как ядро установлено, необходимо сделать его видимым для загрузчика; так, например, если используется загрузчик grub, следует вызвать `update-grub`.

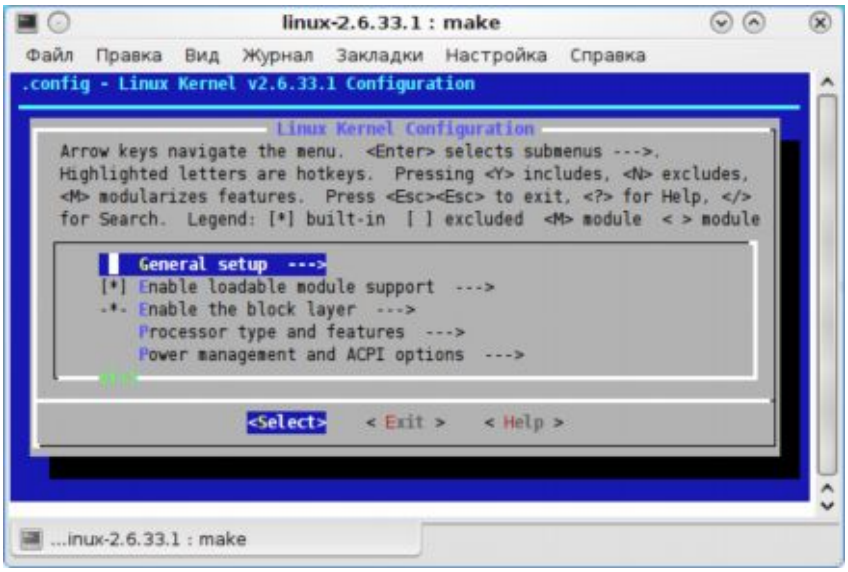


Рис. 6.2.5. `make xconfig` – графический режим.

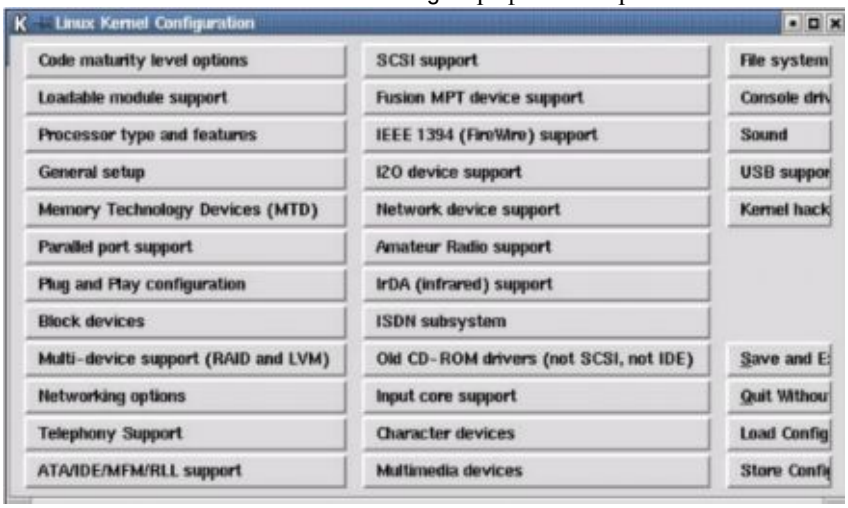


Рис. 6.2.6.

6.3. Обзор важных подсистем MeeGo

Как уже упоминалось в предыдущих лекциях, для разработки пользовательских приложений под ОС MeeGo используется прежде всего фреймворк Qt. В процессе разработки пользовательских приложений, программисту чаще всего не приходится напрямую взаимодействовать с низкоуровневыми подсистемами ОС, но очень важно иметь базовое представление об их работе. Итак, рассмотрим несколько важных подсистем MeeGo:

- *D-bus*

D-Bus — это система межпроцессного взаимодействия, которая позволяет приложениям в операционной системе взаимодействовать друг с другом. D-Bus является частью проекта freedesktop.org. Она обладает высокой скоростью работы, не зависит от рабочей среды, работает на POSIX-совместимых операционных системах. Состоит из двух частей: демона и низкоуровнего API.

Существуют высокоуровневые абстракции над D-Bus для фреймворков Qt, Java, GLib, C#, Python и библиотека для C++.

D-Bus предоставляет системе несколько шин. Системная шина создаётся при старте демона D-Bus и с ее помощью происходит взаимодействие различных демонов. Сессионная шина создается для пользователя, авторизировавшегося в системе. Для каждой такой шины запускается отдельная копия демона, посредством неё будут общаться приложения, с которыми работает пользователь.

Каждое сообщение D-Bus, передаваемое по шине, имеет своего отправителя и своего получателя, их адреса называются путями объектов, поскольку D-Bus предполагает, что каждое приложение состоит из набора объектов, а сообщения пересылаются не между приложениями, а между объектами этих самых приложений.

D-Bus также предусматривает концепцию сервисов. Сервис — уникальное местоположение приложения на шине. При запуске приложение регистрирует один или несколько сервисов, которыми оно будет владеть до тех пор, пока самостоятельно не освободит их. До этого момента никакое другое приложение, претендующее на тот же сервис, занять его не сможет.

Сервисы делают доступной ещё одну функцию — запуск необходимых приложений в случае поступления сообщений для них. Для этого должна быть включена автоактивация, а в конфигурации D-Bus за этим сервисом должно быть закреплено одно приложение. Тогда D-Bus сможет его запустить при появлении сообщения.

После подключения к шине приложение должно указать, какие сообщения оно желает получать, путём добавления масок совпадений (matchers). Маски представляют собой наборы правил для сообщений, которые будут доставляться приложению, фильтрация может основываться на интерфейсах, путях объектов и методах. Таким образом, приложения будут получать только то, что им необходимо; проблемы доставки в этом случае берет на себя D-Bus.

Сообщения в D-Bus бывают четырёх видов: вызовы методов, результаты вызовов методов, сигналы и ошибки. Первые предназначены для выполнения методов над объектами, подключенными к D-Bus; посылая такое сообщение, вы выдаете задание объекту, а он после обработки обязан возвратить вам либо результат вызова, либо ошибку через сообщения соответствующих типов. Сигнальные же сообщения, как им и полагается, ничуть не заботятся о том, что и как делается объектами, они вольны воспринимать их как угодно.

- *Графическая оболочка MeeGo*

Графическая оболочка представляет собой, в целом, оконный менеджер и набор базовых графических приложений. В ОС MeeGo графическая оболочка представлена в трех вариантах, имеющих значительные различия: IVI Meego, Netbook uX и Handset uX. Фреймворк Qt предоставляет единый интерфейс, который позволяет создавать приложения, способные работать в любом из этих вариантов графической оболочки. Тем не менее следует отметить, что для создания приложений хорошо интегрированных в графическую оболочку версии Handset uX необходимо использовать фреймворк meegotouch. Необходимо учесть, однако, что интерфейс meegotouch не входит в Core API MeeGo и может быть подвержен значительным изменениям.

Опишем в общих чертах устройство графической оболчки версий Handset и Netbook uX. Графическая оболочка Netbook uX очень похожа на графическую оболочку ОС Moblin. В качестве оконного менеджера, как и в Moblin, используется mutter. Netbook uX поддерживает библиотеки GTK и clutter, позволяющую использовать возможность акселерации двумерной графики.

В Handset uX функции оконного менеджера исполняет mcomposite. За стартовый экран графической оболочки отвечает meegotouchhome.

6.4. Лабораторная работа № 4 «Использование библиотеки элементов графического интерфейса Qt»

6.4.1. Цель лабораторной работы

Научиться использовать библиотеку элементов графического интерфейса Qt.

6.4.2. Введение

План

- простейшее графическое приложение на Qt
- работа с компоновщиками
- создание приложения ColorViewer
- использование QFileDialog — создание простейшего обозревателя текста

Необходимые знания и навыки

- Знакомство с материалом лаб. работ №№ 2, 3
- Базовое знание языка программирования C++
- Базовое знакомство с фреймворком Qt и механизмом сигналов и слотов (см. лаб. работу №3)
- Базовое знакомство с основными служебными программами Linux (ls, rm, mkdir и т. п.) и принципами работы систем управления пакетами

Необходимые программные и аппаратные средства

- ПК под ОС Linux (поддерживаются дистрибутивы Fedora 13, Ubuntu 10.04, openSUSE 11.3)

6.4.3. Инструкция по выполнению лабораторной работы

Простейшее GUI-приложение на Qt

Рассмотрим следующий фрагмент кода, представляющий простейшее GUI-приложение, созданное с использованием элементов Qt.

```
#include <QApplication>
#include <QWidget>

int main (int argc,
        char **argv)
{
   QApplication app(argc,
                argv);
   QWidget widget(0);

   widget.show();
   return app.exec();
   return 0;
}
```

В этом примере используются два фундаментальных Qt-класса:

- **QApplication** — это движок Qt-приложения и должен создаваться в единственном экземпляре в каждом графическом Qt-приложении. В консольных приложениях используется QCoreApplication. В QApplication запускается диспетчер сигналов и устанавливаются некоторые общие настройки приложения.
- **QWidget** — базовый класс для всех элементов графического интерфейса (виджетов) в Qt, начиная с кнопок и кончая сложными диалогами. Конструктор QWidget может принимать в качестве

аргумента указатель на родительский QWidget. В случае, если передаётся «0», как в настоящем примере, виджет создаётся как самостоятельное окно в системе.

Итак, собрав и запустив пример в каталоге lab04/01, мы обнаружим, что было отображено пустое окно.

Компоновщики (Layout managers)

Мотивация использования. Следующий пример наглядно демонстрирует потребность в компоновщиках.

- Попробуйте добавить в корневой виджет в предыдущем примере несколько элементов типов QPushButton, QLabel, QTextEdit
  - включите соответствующие заголовочные файлы, например,

```
#include <QPushButton>
```

  - создайте объекты, передав в конструкторе указатель на родительский widget

```
QPushButton but1(&widget)
```

- Соберите и запустите приложение.

  Обратите внимание, что все элементы были помещены в левый верхний угол.

Разумеется, все созданные нами дочерние виджеты, могут быть размещены в необходимых местах явно, при помощи задания координат и размеров, но такой метод в крайней степени неудобен и вынудит нас постоянно отслеживать изменения в размере родительского виджета, дабы перекомпоновать дочерние. Компоновщик, представленный общим классом `QLayout`, позволяет избежать этих проблем.

Работа с компоновщиками. Компоновщик отвечает за размещение виджетов в области компоновки в соответствии с некоторыми правилами компоновки. Изменение размеров области компоновки приводит обычно к перекомпоновке.

Рассмотрим виды простейших компоновщиков.

- QHBoxLayout, QVBoxLayout — размещает элементы в один ряд (вертикальный либо горизонтальный)
- QGridLayout — размещает элементы в ячейки таблицы

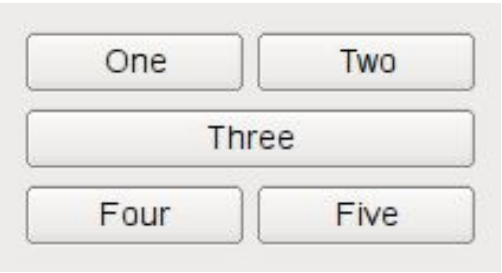


Рис. 6.4.1.

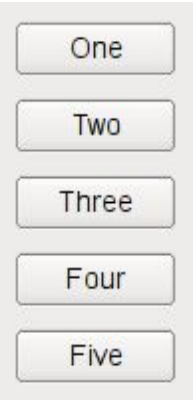


Рис. 6.4.2.

- QFormLayout — размещает элементы сверху вниз в две колонки. Такая организация интерфейса часто используется при заполнении различных форм, где одна колонка — описание, а другая — поле ввода)

Рис. 6.4.3.

Пример работы с компоновщиком

Рис. 6.4.4.

Задание

- пользуясь примером в каталоге lab04/02, создайте приложение с графическим интерфейсом, аналогичным представленному сверху
- используйте классы `QLabel`, `QSpinBox`, `QSlider`, `QPlainTextEdit`.

Знакомство с элементами интерфейса: добавим функциональность.

- Добавим функциональность созданному на предыдущем этапе приложению:
  - спин-боксы и слайдеры будут перемещаться синхронизировано в диапазоне значений от 0 до 255.
  - Цвет фона `QPlainTextEdit` будет меняться соответственно
- Выполнение:
  - Выставляем диапазон допустимых значений для `QSpinBox` и `QSlider` при помощи методов `setMinimum()` и `setMaximum()`
  - запрещаем ввод в текстовое поле: `setEnabled(false)`
  - Реализуем метод `setColor()` и слоты `setRed(int)`, `setGreen(int)`, `setBlue(int)`
  - к слотам подключаем сигналы `QSlider::sliderMoved()` и `QSpinBox::valueChanged()`
  - в реализации слотов синхронизируем значения слайдера и спин-бокса и вызываем `setColor()`
- Для изменения цвета фона текстового поля воспользуемся таблицами стилей для описания стиля элементов.
  - таблицы стилей используют синтаксис CSS
  - будем задавать цвет в виде строки типа `#rrggbb`
  - таким образом, надо задать `QPlainTextEdit` следующий стиль:
  - `QPlainTextEdit { background: #rrggbb; }`
  - задаём стиль при помощи метода `setStyleSheet()` (таблица стиля передаётся в виде строки).

Диалоги

Библиотека графических элементов Qt предлагает набор из нескольких полнофункциональных диалоговых окон, позволяющих выполнять некоторые стандартные операции. Среди них такие, как выбор файла в файловой системе, выбор шрифта, выбор цвета, диалог печати и некоторые другие. Мы рассмотрим в настоящей работе пример использования диалога выбора файла — `QFileDialog`.

Создаём простейший обозреватель текстовых файлов.

- Создайте новый виджет и поместите на него элемент `QTextEdit`.
- Добавьте кнопку QPushButton и подключите её сигнал `clicked()` к слоту `openFile()`

- Реализуйте в слоте выбор имени файла пользователем: `QFileDialog::getOpenFileName()`
- Откройте `QFile` в соответствии с выбранным названием
- Прочитайте его содержимое и поместите в виде текста в элемент `QTextEdit`

6.5. Выводы

В этой лекции мы обсудили основные компоненты ядра Linux и ту функциональность, которую они выполняют. Далее, мы пошагово рассмотрели, как компилируется ядро Linux в простейшем случае. Для более тонких настроек ядра необходимо обратиться к документации по ядру. Затем мы познакомились с важными подсистемами ОС MeeGo, такими как D-Bus и графическая оболочка. В лабораторной работе были рассмотрены примеры использования графических объектов в оболочке Qt.

6.6. Контрольные вопросы

1) Чем, согласно строгому определению, является Linux?
   1. операционной системой
   2. ядром операционной системы
   3. набором служебных программ, созданным в рамках проекта GNU
   4. стандартом, описывающим архитектуру большого семейства ОС
2) Разработчики прикладных программ обычно обращаются к ядру ОС средствами стандартной библиотеки, а не через System Call Interface. Чем это вызвано?
   1. использование SCI усложняет код приложения и делает его хуже переносимым
   2. пользовательские приложения не имеют достаточно прав, чтоб использовать напрямую SCI
   3. спецификация SCI является закрытой
   4. обратиться к функциям SCI можно только на языке assembly
3) В каком пространстве выполняются пользовательские приложения в Linux?
   1. в одном адресном пространстве с ядром ОС
   2. в отдельном адресном пространстве и контролируются ядром ОС
   3. могут выполняться как в адресном пространстве ядра, так и в отдельном адресном пространстве
   4. т. к. Linux не поддерживает виртуальную память, и пользовательские приложения, и ядро выполняются в пространстве физических адресов.
4) Что такое SCI?
   1. интерфейс, посредством которого пользовательские приложения взаимодействуют с ядром Linux
   2. интерфейс, посредством которого ядро Linux взаимодействует с драйверами устройств.
   3. графический пользовательский интерфейс, посредством которого пользователь Linux может выполнять тонкую настройку ядра ОС
   4. ежемесячный журнал, посвящённый проблемам разработки ядра Linux
5) Что такое glibc?
   1. компилятор языка C, используемый для компиляции кода ядра Linux
   2. интерфейс для взаимодействия пользовательских приложений с ядром Linux
   3. стандартная библиотека языка C
   4. библиотека, предоставляющая драйверам устройств в Linux функции для взаимодействия с ядром ОС
6) Кто и когда начал работу над ядром Linux?
   1. Билл Гейтс, 1992
   2. Ричард Столлман, 1985
   3. Линус Торвальдс, 1987
   4. Линус Торвальдс, 1991
7) На каком языке программирования преимущественно ведётся разработка Linux?
   1. C++
   2. C

3. Assembler
4. Fortran

8) Какую архитектуру имеет ядро Linux?
   1. архитектуру, основанную на микроядрах
   2. монолитное ядро
   3. гибридное ядро
9) Что такое монолитное ядро?
   1. части ядра работают в разных пространствах и обращаются друг к другу через интерфейс SCI
   2. модули динамически загружаются в разные адресные пространства
   3. все части ядра работают в одном адресном пространстве
   4. такой архитектуры ядра не существует
10) Сколько строк кода содержал первый релиз Linux?
   1. 10 239
   2. 100 239
   3. 1 239
   4. 2 390
11) Какую аппаратную платформу поддерживал первый релиз Linux?
   1. PowerPC
   2. Sun SPARC
   3. x86-64
   4. i386
12) Кем на момент начала работы над Linux являлся Линус Торвальдс?
   1. штатным разработчиком компании Bell Lans
   2. штатным разработчиком компании Red Hat
   3. студентом университета Хельсинки
   4. безработным
   5. Линус Торвальдс — вымышленный персонаж, "талисман" разработчиков ядра Linux
13) Как принято называть ядро основной ветки?
   1. cinammon
   2. sugar
   3. vanilla
   4. Sodium glutamate
14) По какой лицензии распространяется Linux?
   1. BSD-style
   2. GPL
   3. LGPL
   4. проприетарной
15) Начиная с какой версии, Linux распространяется по лицензии GPL?
   1. 0.99
   2. 2.1
   3. 2.4
   4. 1.1
16) Кто *может* принять участие в разработке ядра Linux?
   1. сотрудники организации Linux Foundation
   2. программисты, сдавшую специальную сертификацию
   3. любой желающий
   4. граждане США
17) Что является основным средством общения для разработчиков Linux?
   1. специализированный форум
   2. канал в IRC
   3. почтовая рассылка
   4. социальная сеть
18) Кто *руководит* разработкой основной ветки ядра Linux?

1. совет из наиболее уважаемых разработчиков Linux
2. Линус Торвальдс
3. ключевые решения по разработке ядра принимаются общим голосованием разработчиков
4. руководитель выбирается раз в четыре года разработчиками ядра

19) Какая система версионного контроля используется при разработке Linux?
1. SVN
2. CVS
3. git
4. BitKeeper

20) Что из списка не является подсистемой ядра?
1. интерфейс системных вызовов
2. управление процессами
3. управление памятью
4. виртуальная ФС
5. все, что указано, относится к подсистемам ядра

21) Что такое VFS (виртуальная файловая система)?
1. дисковая файловая система, используемая по умолчанию в Linux
2. так по лицензионным соображением называется реализация драйвера файловой системы NTFS для Linux
3. специальная абстрактная файловая система, при помощи которой пользовательские программы могут изменять настройки времени исполнения ядра Linux
4. абстракция файловой системы, предоставляющая единый интерфейс доступа к различным файловым системам
5. прослойка, позволяющая драйверам различных файловых систем взаимодействовать с устройствами блочного ввода/вывода

22) В каком адресном пространстве выполняются драйверы устройств?
1. в адресном пространстве пользовательских приложений
2. могут выполняться как в адресном пространстве пользовательских приложений, так и в адресном пространстве ядра
3. в адресном пространстве ядра, так как ядро монолитное

23) Драйверы компилируются в отельные модули с расширением .ko. Что из указанного неверно?
1. это делается для сокращения размера образа ядра
2. это делается для того, чтобы избежать перекомпиляции ядра при изменении драйвера
3. это делается потому, что драйверы исполняются отдельно как процессы в пользовательском пространстве

24) При помощи какой из следующих команд можно подключить модуль драйвера к ядру?
1. insmod <название модуля>
2. lsmod <название модяуля>
3. rmmod <название модяуля>
4. depmod <название модяуля>

25) Какой компилятор используется для сборки ядра?
1. Visual C Compiler
2. gcc
3. g++
4. javac

26) Какая система управления сборкой используется при сборке ядра?
1. qmake
2. GNU make
3. cmake
4. ant

27) Что, применительно к ядру Linux, хранится в файле .config?
1. настройки ядра времени компиляции
2. настройки, с которыми будет запущено ядро

3. настройки для доступа к системе версионного контроля
4. настройщики загрузчика (bootloader) ОС

28) Что такое D-Bus?
1. система управления памятью
2. интерфейс для обмена данными между процессами в Linux
3. интерфейс для обмена данными между процессами в MeeGo
4. интерфейс для взаимодействия с периферийными устройствами

29) Сообщения каких типов встречаются в D-Bus
1. сигналы, ошибки, отчёты
2. вызовы методов, результаты вызова методов
3. вызовы методов, ошибки, отчёты
4. вызовы методов, результаты вызова методов, сигналы, ошибки
5. вызовы методов, результаты вызова методов, сигналы, отчёты

30) Что из указанного про D-Bus неверно?
1. сообщения доставляются посредством шин
2. приложение, использующее D-Bus, может зарегистрироваться как сервис и указать, какие сообщения оно будет получать
3. приложения регистрируют деревья объектов, каждому из объектов присваивается уникальное имя
4. объекты реализуют методы, которые могут быть вызваны при помощи отправки соответствующего сообщения
5. приложение, которому адресуется сообщение, должно быть предварительно запущено

31) В рамках какого проекта разрабатывается D-Bus?
1. GNU
2. freedesktop
3. GNOME
4. KDE

32) В каком из вариантов графической оболочки MeeGo поддерживается библиотека clutter?
1. Netbook uX
2. Handset uX
3. не поддерживается нигде
4. поддерживается во всех

33) Какой оконный менеджер используется в Handset uX?
1. mutter
2. Metacity
3. Compiz
4. mcomposite

34) Какие виджет-фреймворки можно использовать для создания приложений для MeeGo Handset uX?
1. meegotouch
2. Qt
3. meegotouch и Qt
4. GTK
5. GTK и Qt

35) Какой оконный менеджер используется в Handset uX?
1. mutter
2. metacity
3. twm
4. mcomposite

36) Какое приложение отвечает за показ стартового экрана в Handset uX?
1. mutter
2. clutter
3. meegotouch
4. meegotouchhome

Список литературы

Ядро Linux

1. http://en.wikipedia.org/wiki/Linux_kernel
2. http://kernelnewbies.org/
3. http://kernel.org/doc/man-pages/
4. Книга Greg Kroah-Hartman «Linux Kernel in a Nutshell» http://www.kroah.com/lkn/

D-Bus

5. http://www.freedesktop.org/wiki/Software/dbus.

Документация по Qt

6. http://doc.qt.nokia.com/4.7/index.html
7. http://doc.qt.nokia.com/4.7/widgets-and-layouts.html
8. http://doc.qt.nokia.com/4.7/layout.html

Раздел 2. Библиотеки промежуточного слоя ОС MeeGo

7. MeeGo API: сервисы коммуникации

MeeGo API: Comms Services. Connection Mgmt [ConnMan], Telephony [oFono], VOIP, IM, Pres. [Telepathy], Bluetooth [BlueZ]. Примеры взаимодействия.

7.1. Введение

Этой лекцией мы открываем новый раздел курса, посвященный подсистемам, которые формируют промежуточный слой ПО для мобильных устройств. Основной целью этой части курса будет разобрать основные подсистемы, находящиеся на промежуточном уровне (слое программного обеспечения, находящегося между низкоуровневым ПО типа ядра ОС и пользовательским ПО), обеспечивающие его основную функциональность.

Начнем с подсистем, отвечающих 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.

7.2. Сервисы коммуникации

Communications services в ОС MeeGo в настоящее время представлены следующими подсистемами:

- ***стека телефонии oFono***

oFono – это новейшая разработка компаний Intel и Nokia. OFono представляет собой системный сервис и простой программный интерфейс, предоставляющий пользовательским приложениям доступ к функциям сотовой телефонии. Он ориентирован прежде всего на разработку программ для работы на смартфонах, поддерживающих стандарты GSM или UMTS. Разработчикам конечных приложений oFono позволит полностью сосредоточится на создании пользовательского интерфейса, без углубленного изучения тонкостей работы телефонии. В качестве API задействован уже упоминавшийся в предыдущих лекциях D-Bus. Каких-либо ограничений на выбор языков программирования не накладывается. Особым преимуществом стека oFono является универсальность программного интерфейса, достаточно простого для изучения, который не зависит от используемого аппаратного обеспечения.

- ***диспетчера соединений ConnMan***

программа-демон для управления Интернет-соединениями в мобильных устройствах под управлением ОС семейства Linux. Модульная система, легко расширяемая плагинами для поддержки всех видов проводных и беспроводных соединений.

- ***стека Bluetooth (BlueZ)***

Bluez – стек технологии Bluetooth для Linux. Его цель — реализация спецификаций стандартов технологии Bluetooth для Linux. Стек BlueZ поддерживает все основные протоколы и профили Bluetooth. Был первоначально разработан компанией Qualcomm. Поддерживает ядра Linux версии 2.4.6 и выше.

- ***фреймворка Telepathy***

Telepathy – это гибкий модульный фреймворк связи, ориентированный на общение в режиме реального времени как посредством обмена текстовыми сообщениями, так и при помощи голосовой и видеосвязи. Этот сервис может использоваться несколькими клиентскими приложениями одновременно. В то время, как для пользовательских приложений Telepathy предоставляет единый интерфейс, использующий D-Bus, его back end имеет поддержку для самых популярных протоколов, включая: Jabber/XMPP/Google Talk/Jingle, SIP, MSN, Yahoo/AIM и IRC. Мы обсудим также модуль telepathy-ring, интегрирующий функционал сотовой связи в фреймворк telepathy.

В предыдущих лекциях упоминалась, что фреймворк Qt рекомендуется в качестве основного средства разработки для ОС MeeGo. В настоящей лекции мы рассмотрим интеграцию упомянутых выше подсистем с Qt, а также изучим возможности модуля Qt Mobility для разработки приложений, использующих функции связи.

7.3. Обзор реализаций сотовой связи в смартфонах

Прежде чем мы начнем говорить о стеке телефонии oFono, хочется в общих чертах понять, как устроен смартфон, и каким образом программное обеспечение, которое на нём выполняется, способно использовать функционал сотовой связи (например, GSM или 3G).

Может показаться, что операционная система смартфона получает данные непосредственно из радиоэфира и непосредственно с ними и работает. В действительности это не так. Вся логика работы со стандартами связи вынесена в отдельный аппаратный модуль. Во всех пользовательских устройствах существует процессор общего назначения, на котором исполняются ядро ОС смартфона, служебные и пользовательские приложения. Существующий отдельно модуль сотовой связи фактически полностью реализует протокол сотовой связи, отвечает за кодирование и декодирование сигнала, за работу с SIM-картой и т. д. Обмен данными между ним и процессором общего назначения происходит при помощи команд AT (стандарт 3GPP 27.007). AT представляет из себя старый текстовый протокол управляющих команд. Он является стандартом де-факто для управления модемами и, в частности, сотовыми модемами. Фактически все операции сотовой связи, такие как совершение звонков, установка GPRS / 3G соединений, прием и отправка текстовых сообщений, регистрация в сети, чтение контактов с SIM-карты, осуществляются посредством отправки соответствующих AT команд. Когда мы совершаем вызов, приложение-наборщик (dialer) всего лишь отправляет AT команду сотовому модулю и настраивает звуковую подсистему устройства, соединяя её с входами и выходами сотового модуля. Все дальнейшие действия по установлению соединения, кодированию и декодированию звукового сигнала, по модуляции радиосигнала выполняется модулем сотовой связи, в то время как процессор общего назначения находится практически в бездействии.

Набор команд AT несколько устарел и вызывает ряд определенных проблем. Различные производители модемов всячески расширяют его для того, чтобы обеспечить необходимую функциональность, что отрицательно сказывается на совместимости протоколов. Кроме того, модули сотовой связи, выпускаемые компанией Nokia, используют вовсе иной протокол управления — ISI.

7.4. Стек телефонии oFono

Когда мы говорим о стеке телефонии, то подразумеваем какое-то программное обеспечение, которое будет запускаться в операционной системе смартфона, и будет обеспечивать пользовательским приложениям доступ к функционалу сотовой телефонии и возможность взаимодействия с некоторыми подсистемами телефона. Необходимость такого программного слоя как oFono обусловлена тем, что в его отсутствие взаимодействие пользовательских приложений с функциями телефонии существенно усложняется. Приложения вынуждены использовать сложный и неуклюжий протокол AT команд для управления сотовым модулем, при этом учитывая, что используемое аппаратное обеспечение может использовать какую-то модификацию протокола или его устаревшую версию; это делает такие приложения плохо переносимыми и сложными для понимания. oFono решает эти проблемы, предоставляя единый высокоуровневый интерфейс к функциям сотовой телефонии.

oFono – это уже не первая попытка реализации стека телефонии в ОС Linux. В рамках таких проектов, как Maemo и Android, создавших операционные системы для смартфонов на базе Linux, был создан в том числе и стек сотовой телефонии. Можно выделить три основные попытки, кроме oFono, создать стек телефонии — это проекты Qtopia, freesmartphone и стек телефонии Android. Эти проекты по различным причинам не подходили для использования в MeeGo, в свете чего было принято решение развивать проект oFono и интегрировать его в качестве стека телефонии.

oFono – это уровень абстракции над модулем сотовой связи, скрывающий за собой как сложный протокол взаимодействия с аппаратным обеспечением. так и особенности использования стандартов сотовой связи. Ведь даже передача простого текстового сообщения по сотовой связи сопряжена с выполнением весьма непростого кодирования текста, его упаковкой и отправкой сложной управляющей команды. При этом непосредственно ядро oFono не зависит от конкретной платформы. Для взаимодействия с пользовательскими приложениями используется интерфейс D-Bus.
oFono состоит из следующих основных компонент:

1. Демон-диспетчер. Демонами в Unix-подобных системах называют фоновые системные процессы, не взаимодействующие напрямую с пользователями. Обычно они представляют некоторую системную службу. Демон oFono отвечает за такие задачи, как определение и подключение нового оборудования.
2. Атомы, которые представляют собой абстракцию некой базовой функциональности, реализованной ядром системы. Так, существует атом, например, для отправки сообщений. Атомы могут регистрироваться в D-Bus для предоставления сервиса.
3. Плагины, среди которых можно выделить:
   - драйверы атомов, представляющие собой абстракции нижнего уровня и определяющие реализацию функционала атома в элементарных действиях.
   - драйеверы модемов, являющиеся абстракцией над модулем сотовой связи.
   - плагины, реализующие дополнительную функциональность — такую, как, например, работу с историей звонков и сообщений.

   Функциональность, поддерживаемая oFono включает в себя:

– голосовые звонки, в том числе тонкие настройке по работе с ними. oFono поддерживает такие функциональности как ожидание вызова, конференц-звонки, выбор используемых звуковых кодеков.
– отправку и прием текстовых сообщений
– GPRS (возможно лишь одно активное соединение)
– обнаружение сети и регистрация в ней
– работу с PIN-кодами
– чтение контактов с SIM-карты
– прием широковещательных сообщений от соты – cell broadcast
– USSD-запросы .

oFono не поддерживает WAP, MMS, EMS, видеозвонки, сохранение контактов в память SIM .

Отсутствие поддержки MMS обусловлено тем, что логически механизм отправки и приема MMS находится выше чем реализован стек oFono. Технологии WAP и сохранение контактов в память SIM были исключены, так как считаются устаревшими. Поддержка видеозвонков не была реализована на данном этапе, так как, согласно данным мировых сотовых операторов, эта услуга сравнительно редко используется; в последующих версиях, впрочем, эта функциональность может быть поддержана.

7.5. Диспетчер соединений ConnMan

ConnMan – это демон для управления Интернет-соединениями в Linux. ConnMan легко интегрируется в программное обеспечение мобильного устройства. В современных портативных устройствах для подключения к сети используются самые различные каналы, например: проводное соединение, Wi-Fi, сотовая связь. Для выхода в сеть может использоваться также канал Bluetooth. Изначально ConnMan был разработан именно для мобильных решений и поэтому не требователен к ресурсам. О широком распространении ConnMan свидетельствует, в частности, то, что в последнем релизе Ubuntu Netbook Edition ConnMan заменил использовавшийся для этой цели ранее NetworkManager (Рис. 7.5.1).

Рис. 7.5.1.

Интерфейс ConnMan реализован на основе D-Bus. ConnMan поддерживает сетевые соединения с использованием следующих провайдеров:

– Ethernet
– Wi-Fi
– Wi-Max
– GSM / UTMS / 3G
– Bluetooth (выход в сеть через другое устройство).

ConnMan поддерживает сортировку доступных соединений, автоматическое подключение к ним, а также возможность выбора соединения пользователем.

7.6. Bluetooth и реализация BlueZ

Bluetooth – это открытая спецификация беспроводных сетей для передачи данных на короткие расстояния. Bluetooth обеспечивает обмен информацией между такими устройствами как карманные и обычные персональные компьютеры, мобильные телефоны, ноутбуки, принтеры, цифровые фотоаппараты, мышки, клавиатуры, джойстики, наушники, гарнитуры на надёжной, недорогой, повсеместно доступной радиочастоте для ближней связи. Максимальное расстояние между устройствами находится в диапазоне от 10 до 100 м в зависимости от используемой версии протокола. Bluetooth определяет ряд профилей, сценариев использования канала, для различных целей. Сюда входит и профиль для передачи аудиопотока, и для пересылки изображений, и для управлениями стандартными функциями телевизора, и многие другие.

Наиболее распространенной реализацией стека Bluetooth для Linux является BlueZ. Bluez поддерживает версию стандарта 2.1 + EDR (увеличенная скорость передачи данных). Также в нем реализовано большнство существующих профилей Bluetooth.

7.7. Диспетчер соединений – ConnMan

После того как мы отдельно обсудили, что представляют из себя технологии oFono, ConnMan и Bluez, рассмотрим, как они могут взаимодействовать между собой и реализовывать общую систему связи в мобильном устройстве. На Рис. 7.5.2 показан пример подобной реализации:

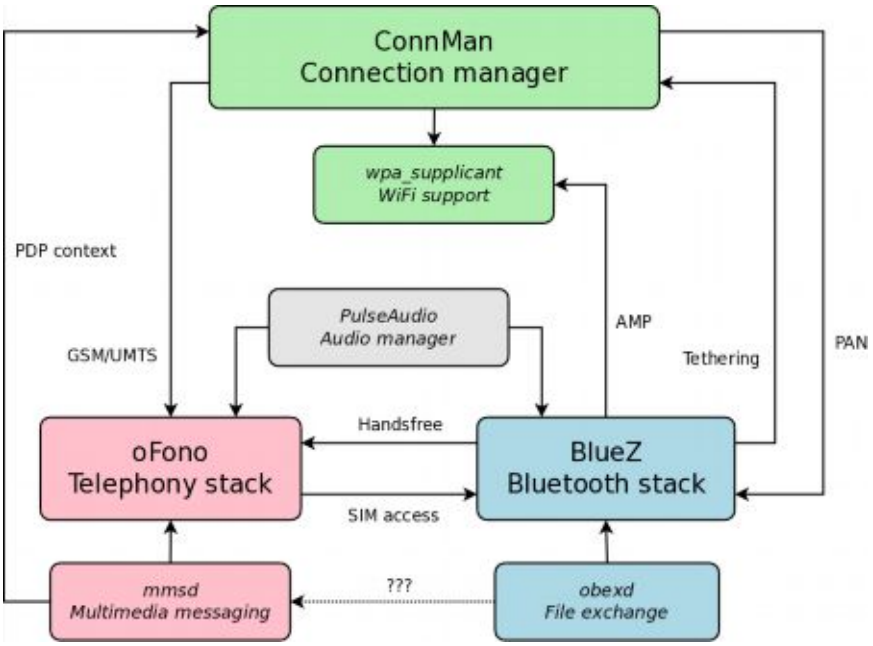
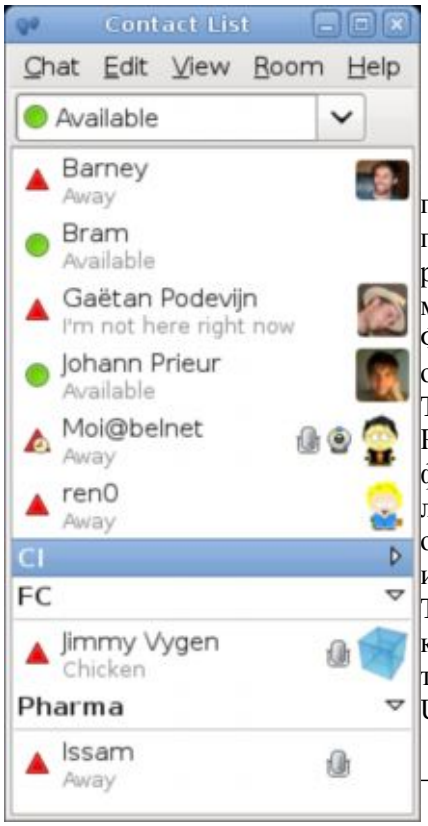


Рис. 7.5.2.

Рассматриваемый пример не покрывает все возможности взаимодействия oFono, Connman и Bluez, но показывает возможный сценарий их использования.

На Рис. 7.5.2 показано, в частности, что oFono взаимодействует со стеком Bluez при помощи реализованного для него плагина HandsFree, предоставляя, таким образом, функциональность устройства громкой связи. Диспетчер соединений ConnMan имеет возможность использовать в качестве провайдеров сетевого соединения Bluez и oFono. В качестве связующей шины и высокоуровневого API для использования функций телефонии в программах oFono и ConnMan используется D-Bus, что позволяет легко налаживать их взаимодействие.



7.8. Универсальный коммуникационный фреймворк Telepathy

Telepathy – это универсальный фреймворк для создания приложений для общения. Основной целью его создания было предоставить возможность настольным приложениям таким, как редакторы текста или игры, интегрироваться с чатами, системами мгновенного обмена сообщениями и другими средствами общения. Фактически back end систем связи выносится в отдельный сервис и общение с ним происходит через интерфейс D-Bus.

Telepathy разрабатывается в рамках проекта Freedesktop.org. Разработчики Telepathy надеются, что поддерживаемая новая функциональность сможет существенно улучшить качество общения людей и дать возможность для разработки новых приложений, опирающихся на возможность общения. Одним из наиболее известных проектов на сегодня, разработанным под фреймвороком Telepathy, является Empathy, который по умолчанию является клиентом для систем обмена мгновенными сообщениями и IP-телефонии (аудио и видео) в операционных системах Fedora 12 и Ubuntu 9.10.

Фреймворк Telepathy состоит из двух логических компонент:

– набор диспетчеров соединений (connection managers),

поддерживающих большое количество протоколов (таких, как XMPP, Google Talk, AOL Instant Messenger и MSN);

– утилиты управления каналами и аккаунтами.

Telepathy имеет гибкую архитектуру. Поддержка нового протокола может быть добавлена в фреймворк посредством реализации соответствующего диспетчера соединений. Telepathy является сервисом, к которому могут подключаться множественные клиенты . Взаимодействие между ними осуществляется через D-Bus. Передача данных осуществляется по каналам (text channel, streammedia channel).

Одним из наиболее изящных применений Telepathy является интергация функциональности сотовой связи в общую модель систем обмена сообщениями и голосовой связи, реализованная при помощи диспетчера соединений Telepathy-ring. Несложно заметить, что основная функциональность сотовой связи — голосовые звонки и sms-сообщения по сути не отличаются от сообщений в таких сетях как ICQ и MSN, и от голосовых звонков в системах VoIP. Telepathy-ring, таким образом, позволяет использовать ваше любимое приложение как для отправки Jabber-сообщений, так и для отправки sms, что весьма актуально на сегодняшний день, когда любой контакт в записного книги вашего устройства может быть представлен не только номером телефона, но также Jabber ID, номером ICQ и т. д.

Telepathy-ring полностью скрывает за собой работу со стеком oFono.

7.9. Взаимодействие с Qt

Как уже говорилось на вводных лекциях по разработке для ОС MeeGo, фреймворк Qt предлагается разработчикам в качестве основы для любых пользовательских приложений. Компонента QtMobility при этом предоставляет разработчикам общий интерфейс к функциональностям, характерным для мобильных устройств. К сожалению, в настоящий момент в QtMobility не реализована работа с функциями телефонии. Разработчиками MeeGo, впрочем, разрабатывается обертка ofono-qt, предоставляющая Qt совместимый интерфейс к функциям oFono. Пользовательские приложения на Qt могут также использовать компоненту qt-dbus для непосредственного взаимодействия с oFono через шину D-Bus.

Для работы с Telepathy Qt-приложения могут использовать обертку telepathy-qt4, пример использования которой приведен в одной из наших лабораторных работ. Аналогичную обертку для ConnMan предоставляет libconnman-qt.

7.10. Лабораторная работа № 5 «Сервисы коммуникации MeeGo: IM-клиент с использованием telepathy»

7.10.1. Цель лабораторной работы

Научиться использовать в своих приложениях telepathy.

7.10.2. Введение

План

1. Подготовка
- Установка необходимых программных пакетов
- Сборка и установка новейшей версии telepathy-qt4, конфигурация динамического линковщика
- Создание учётной записи jabber

2. telepathy-qt4
- Обзор технологии. основные модули.
- Реализация простого клиента для текстового чата с использованием lowlevel API (в обход ChannelDispatcher)

- Реализация простого клиента для текстового чата с полной инициализацией.

Необходимые знания и навыки
- Знакомство с материалом лаб. работ №№ 2, 3
- Базовое знание языка программирования C++
- Базовое знакомство с фреймворком Qt и механизмом сигналов и слотов (см. лаб. работу №3)
- Базовое знакомство с основными служебными программами Linux (ls, rm, mkdir и т. п.) и принципами работы систем управления пакетами.

Необходимые программные и аппаратные средства
- ПК под ОС Linux (поддерживаются дистрибутивы Fedora 13, Ubuntu 10.04, openSUSE 11.3)
- Интернет-соединение.

7.10.3. Инструкция по выполнению лабораторной работы

7.10.3.1. Подготовка

Установка базовых пакетов

(Указания даны на основе Ubuntu 10.04.).
- Установить (в дополнение к пакетам, установленным в прошлых работах) следующие пакеты при помощи команды apt-get install
  - qt4-sdk
  - telepathy-mission-control-5, telepathy-gabble
  - cmake (необходим для сборки telepathy)

Сборка и установка telepathy-qt4

(Используем версию библиотеки 0.5.1.).
- Ссылка для скачивания пакета исходных файлов: http://telepathy.freedesktop.org/releases/telepathy-qt4/telepathy-qt4-0.5.1.tar.gz
- Распаковываем архив (см. лаб.работу №2)
- Запускаем конфигурацию:

```
cmake
```

- Запускаем сборку:

```
make all
```

- Устанавливаем библиотеку:

```
sudo make install
```

Конфигурация динамического линковщика

Для того, чтоб отделить устанавливаемую «рабочую» версию пакета от пакетов, установленных в системе постоянно, заголовочные файлы и объектный файл (*.so) устанавливаются, соответственно, в каталоги /usr/local/includes и /usr/local/lib, а не в каталоги /usr/includes и /usr/lib, которые обычно используются для этих целей. Это, однако, означает, что при сборке будет необходимо указать правильный префикс каталога, по которому следует искать заголовочные файлы и файл библиотеки. Необходимо также убедиться, что правильный файл *.so находится в кеше динамического линковщика, который отвечает за подключение прилинкованных динамически библиотек во время загрузки приложения.
- Проверьте, находится ли нужная версия библиотеки в кеше динамического линковщика.

```
ldconfig -p | grep /usr/local/lib/libtelepathy-qt4
```

- Если команда не дала какого-либо вывода, добавляем строку "/usr/local/lib" в файл /etc/ld.so.conf.d/usr-local-lib.conf
- Выполняем sudo ldconfig для обновления кеша

Создание учётной записи jabber

Для выполнения лабораторной работы необходимы 2 учётные записи jabber. Следует отметить, что учётная запись GTalk также является учётной записью jabber. Если у вас нет регистрации в jabber, учётную запись jabber.org можно создать бесплатно, пройдя по ссылке: https://register.jabber.org/.

- Зарегистрируйте учётные записи в клиенте jabber (например, в empathy).
- Добавьте в список контактов каждой записи вторую и авторизуйте их, для того, чтоб между этими двумя учётными записями можно было обмениваться сообщениями.

7.10.3.2. Работа с TelepathyQt4

Telepathy является общим back-end'ом для различных сервисов IM, аудио- и видеотелефонии. Объединяя эти сервисы, Telepathy предоставляет единый простой интерфейс, что позволяет клиентским приложениям использовать любой из сервисов без необходимости реализовывать поддержку конкретного протокола. Telepathy позволяет также без каких-либо ухищрений запускать несколько клиентских приложений, использующих один сервис. Для взаимодействия между различными компонентами системы, Telepathy использует шину D-Bus.

Telepathy-qt4 представляет собой прозрачную Qt-обёртку для D-Bus интерфейса, который используется Telepathy. Использование D-Bus накладывает заметный отпечаток на API Telepathy-qt4, который в силу этой причины ялвяется асинхронным. Эта асинхронность реализована при помощи механизма сигналов и слотов: при выполнении D-Bus вызова создаётся объект класса PendingOperation, который по завершении вызова и получении результата отправляет сигнал `finished()`.

Специфика интерфейса заключается также в том, что дабы минимизировать обмен сообщениями по сравнительно медленному каналу D-Bus, Telepathy-qt4 зачастую не загружает полную информацию об D-Bus-объекте и не включает функциональности, которые генерируют частые сообщения. В силу этого часть функциональности некоторых объектов Telepathy-qt4 бывает изначально недоступна. Для того, чтоб активировать те или иные функции, необходимо выполнить запрос `becomeReady()` с соответствующим значением Features.

Во избежание утечек памяти интерфейс Telepathy-qt4 активно использует «умные» указатели.

Базовые компоненты telepathy-qt4

- **ConnectionManager** — представляет сервис, обеспечивающий взаимодействие по определенному протоколу или по группе протоколов. Например, gabble (для протокола jabber)
- **Connection** — соединение, создаваемое сущностью ConnectionManager
- **Channel** — канал обмена данными, привязанный к определенной сущности Connection. Может быть текстовым, медиаканалом и каналом для передачи файлов.
- **AccountManager** — менеджер зарегистрированных в системе аккаунтов
- **Account** — зарегистрированный в системе пользовательский аккаунт
- **AbstractClient** — клиент, обработчик каналов, принадлежащий одному из типов:
  - **AbstractClientApprover**
  - **AbstractClientHandler** — непосредственно обработчик
  - **AbstractClientObserver** — наблюдатель. Может использоваться, например, для журналирования
- **ChannelDispatcher** — определяет, какой обработчик будет обрабатывать созданный канал
- **ClientRegistrar** — регистрирует обработчики.

Низкоуровневая реализация клиента

Низкоуровневый подход, который разработчики Telepathy-qt4 не рекомендуют использовать в обычных приложениях, характерен тем, что не использует сохраненную в системе информацию об

учётных записях и тем, что самостоятельно оперирует каналами, в то время, как обычный подход, который будет описан ниже, полагается в этом на диспетчер каналов.

Рассмотрим пример, который находится в архиве telepathy-client-lowlevel.tar.gz. Пример представляет собой простой чат-клиент для jabber. Осовные этапы его работы таковы:

- Получаем сущность ConnectionManager для gabble

```
ConnectionManager::create("gabble")
```

- Используя low-level интерфейс ConnectionManager, создаём соединение

```
CM->lowlevel()->requestConnection()
```

- Активируем соединение:

```
connection->lowlevel->requestConnect()
```

- Запрашиваем сущность Contact для заданного имени через ContactManager

```
connection->contactManager()->conatctsForIdentifiers()
```

- Используя низкоуровневый интерфейс `Connection`, запрашиваем создание текстового канала (TextChannel)

```
connection->lowlevel()->createChannel()
```

- Запрашиваем активацию необходимой функциональности канала (в особенности `FeatureMessageQueue` для приёма сообщений)

```
channel->becomeReady(QSet<Tp::Feature>)
```

- Канал готов к работе: подключаемся к сигналу `TextChannel::messageReceived()` и вызываем слот `sendMessage()` для отправки сообщений.

Сборка и запуск приложения

- Выполняем для генерации Makefile

```
qmake [-makefile]
```

Внимание: если вами была ранее установлена версия libtelepathy-qt4 из репозитория, то при вызове `make` необходимо указывать

```
SUBLIBS="-L/usr/include"
```

иначе линковщик не сможет найти нужную версию библиотеки.

- Запуск приложения:

```
./client <account_1> <Account_1_password> <account_2>
```

Высокоуровневая реализация клиента

Высокоуровневый подход в API Telepathy-qt4 использует для создания каналов абстракции AccountManager и Account. Вместо обращения к одному из менеджеров соединений и создания соединения с сервисом с заданными параметрами, мы используем Account, учётную запись, зарегистрированную в системе ранее (в нашем примере, однако, мы сами создаём и удаляем её). Для Account можно задать состояние подключения («в сети», «недоступен» и т. п.) и затребовать канал необходимого типа.

Следует отметить, однако, что созданные таким образом каналы не возвращаются непосредственно в результате вызова, а передаются для обработки диспетчеру каналов, который, согласно значению параметра PreferredHandler, передаёт канал тому или иному обработчику. Очевидно, для того, чтоб получить управление над этим каналом, наше клиентское приложение должно зарегистрироваться как обработчик каналов.

Рассмотрим основные шаги работы высокоуровневой реализации клиента, которая находится в архиве telepathy-client.tar.gz:

- Реализуем свой AbstractClientHandler, который будет принимать вновь созданные соединения и создавать для них ChatWidget.
- Регистрируем клиент
- Получаем сущность AccountManager и создаём новый Account
- Активируем аккаунт

```
Account::setEnabled()
```

- Создаём канал текстового чата для заданного пользователя с указанием нашего обработчика в качестве PrefferedHandler

```
Account::ensureTextChat()
```

7.10.4. Задания для самостоятельной работы

Подпункты задания упорядочены в порядке возрастания сложности.

1. Для примера telepathy-client-lowlevel:
   - реализовать поддержку дополнительных протоколов обмена сообщениями (кроме jabber)
   - реализовать вывод времени получения сообщений.
2. Для примера telepathy-client:
   - реализовать отображение имён собеседников в ChatWidget
   - реализовать отображение списка контактов для заданной учетной записи
   - реализовать создание каналов по клику по имени в списке контактов
   - реализовать отображение статусов

7.11. Выводы

В этой лекции мы разобрали подсистему промежуточного слоя мобильного устройства, отвечающую за коммуникацию. Мы обсудили, как она организована, какие компоненты включает и как эти компоненты взаимодействуют между собой. Был дан обзор технологии Telepathy, в котором мы показали, для чего разрабатывалась эта технология и какие у нее есть возможности использования.

7.12. Контрольные вопросы

1) Что такое oFono?
   1. стек телефонии
   2. диспетчер соединений
   3. стек Bluetooth
   4. универсальный коммуникационный фреймворк
2) Какие компании приняли участие в разработке oFono?
   1. Nokia
   2. Intel
   3. Intel и Nokia
   4. Nokia и Google
3) На работу с какими стандартами связи ориентирован oFono?
   1. GSM и UMTS
   2. WiMax
   3. Wi-Fi
   4. RFID
4) Каким образом пользовательские приложения взаимодействуют с oFono?
   1. интерфейс SCI
   2. шина D-Bus
   3. через модуль QtMobility фреймворка Qt
   4. при помощи механизма общего доступа к памяти
5) Выберите неправильное утверждение в списке.
   1. oFono представляет собой абстракцию над модулем сотовой связи
   2. oFono скрывает работу с протоколами GSM
   3. ядро реализации oFono не зависит от используемой аппаратуры
   4. oFono написан с использование фреймворка Qt
6) Какие языки программирования можно испольовать для разработки пользовательских

приложений, взаимодействующих с oFono?
   1. так как в качестве API задействован D-Bus, каких-либо ограничений на выбор языков программирования не накладывается
   2. C/C++
   3. команды AT
   4. Python
7) Какие функциональности не поддерживаются oFono?
   1. видеозвонки, GPRS-соединения, чтение контактов из памяти SIM
   2. WAP, GPRS-соединения, видеозвонки, чтение контактов из памяти SIM и USSD-запросы
   3. WAP, MMS, EMS, видеозвонки, сохранение контактов в память SIM
   4. MMS, EMS, видеозвонки, чтение контактов из памяти SIM
8) Что такое ConnMan?
   1. плагин для oFono
   2. драйвер GSM модема
   3. диспетчер соединений
   4. универсальный коммуникационный фреймворк
9) Основные компоненты oFono это —
   1. менеджеры соединений и MissionControl
   2. демон-диспетчер, атомы, плагины
   3. менеджеры соединений, демон-диспетчер и атомы
   4. атомы, плагины
10) Что такое Bluez?
   1. реализация стека Bluetooth для Linux
   2. реализация стека телефонии для Linux
   3. диспетчер соединений
   4. драйвер для работы с Blueray-устройствами
11) Какую версию стандарта Bluetooth поддерживает Bluez?
   1. 2.1
   2. 2.1 + EDR (увеличенная скорость передачи данных)
   3. 2.0 + EDR
   4. 3.0 + HS
12) Укажите, что из перечисленного неверное отражает взаимодействие oFono и Bluez?
   1. oFono реализует plugin для HandsFree-профиля Bluetooth
   2. BlueZ обращается к oFono для чтения данных с SIM-карты
   3. oFono и Bluez пока не могут взаимодействовать друг с другом
13) Укажите, что из перечисленного верно отражает взаимодействие ConnMan с oFono и Bluez?
   1. ConnMan может использовать BlueZ и oFono для подключения к сети
   2. ConnMan не может использовать BlueZ для подключения к сети, так как не поддерживает Bluetooth соединение
   3. ConnMan не может использовать oFono, так как не работает с D-Bus
14) Что такое Telepathy?
   1. универсальный фреймворк для использования функций определения местоположения и геокодинга
   2. универсальный фреймворк для работы с сервисами обмена мгновенными сообщениями, видео- и аудиотелефонии
   3. универсальный диспетчер соединений
   4. плагин для oFono
15) В рамках какого проекта был разработан Telepathy?
   1. GNU
   2. совместный проект Intel и Nokia
   3. Qt
   4. freedesktop.org
16) В основе какого популярного IM / VOIP клиент написан на Telepathy?

1. Miranda
2. Pidgin
3. Empathy
4. GTalk

17) Какое из следующих утверждений об архитектуре Telepathy неверно?
1. Работа с различными протоколами реализована в Telepathy в модулях Connection Manager.
2. ConnectionManagers динамически прилинкованы к модулю Mission Control.
3. Модуль Mission Control запускается как сервис и регистрируется в D-Bus
4. Пользовательские приложения взаимодействуют с Telepathy, обращаясь к Mission Control

18) Какой интерфейс используется для взаимодействия Telepathy с клиентскими приложениями?
1. механизм сигналов и слотов Qt
2. шина D-Bus
3. механизм сигналов и слотов Qt и шина D-Bus
4. UNIX-сокеты

19) Что представляет из себя telepathy-ring?
1. connection manager, реализующий поддержку GSM-телефонии для Telepathy
2. connection manager, реализующий поддержку VOIP-телефонии для Telepathy
3. центральный компонент в Telepathy, координирующая работу connection managers, и предоставляющая интерфейс для пользовательских приложений
4. почтовая рассылка проекта Telepathy

20) Какие виды каналов существуют в Telepathy?
1. видеоканалы, аудиоканалы
2. видеоканалы, аудиоканалы, текстовые каналы
3. текстовые каналы, каналы потоковых данных
4. текстовые каналы, каналы потоковых данных, каналы управления

21) Какой программный продукт из перичисленных ниже используется в telepathy-ring?
1. ConnMann
2. oFono
3. Telepathy
4. Bluez

22) Какой программный компонент отвечает за кодирование и декодирование звука при совершении голосовых звонков в смартфонах на базе MeeGo.
1. драйвер модема oFono
2. демон-диспетчер oFono
3. PulseAudio
4. никакой, за это отвечает аппаратный сотовый модуль

23) Какой протокол обычно используется для управления сотовым модемом?
1. TCP
2. UDP
3. D-Bus
4. команды AT

24) Каким образом приложения на Qt могут использовать функциональность фреймворка Telepathy?
1. при помощи библиотеки QDBus
2. при помощи обёртки telepathy-qt4
3. при помощи библиотеки QDBus и обёртки telepathy-qt4
4. фреймворк Telepathy не может быть использован в Qt-приложениях

25) Что из перечисленного ниже не относится к преимуществам фреймворка Telepathy?
1. Telepathy предоставляет единый интерфейс к различным сервисам
2. поддержка новых сервисов/протоколов реализуется в отдельных модулях и не затрагивает ядро фреймворка и модули других сервисов
3. Telepathy поддерживает наибольшее количество сервисов и протоколов.
4. Telepathy могут одновременно несколько клиентских приложений

Список литературы

1. oFono — документация по проекту. Исходные тексты доступны по адресу
http://meego.gitorious.org/meego-cellular/ofono
2. wiki проекта Telepathy
http://telepathy.freedesktop.org/wiki/
3. Документация по TelepathyQt4
http://telepathy.freedesktop.org/doc/telepathy-qt4/
4. Bluetooth
http://en.wikipedia.org/wiki/Bluetooth

8. MeeGo API: сервисы Интернета и местоположения

MeeGo API: Internet Services. Layout Engine [WebKit], Web Services [libSocialWeb], Location [GeoClue]. Пример связи двух устройств по Wi-Fi и через Интернет для передачи данных от GPS приемника одного из устройств.

8.1. Введение

Эта лекция посвящена подсистеме ОС MeeGo, которая отвечает за работу с интернетом и за определение местоположения. Мы дадим краткий обзор браузерного движка WebKit и его роли в MeeGo, библиотеки libsocialweb, использующейся для взаимодействия с социальными сетями, обсудим различные способы определения местоположения, применяемые на сегодняшний день в мобильных устройствах и возможность их использования в приложениях для MeeGo.

8.2. Браузерный движок WebKit

8.2.1. Основная функциональность браузерного движка

Браузерный движок (layout engine, rendering engine) — это программная компонента, лежащая в основе браузера. Это вынесенная в отдельный программный модуль основная функциональность браузера, которая заключается в преображении веб-страницы, представленной совокупностью HTML- и XML-файлами, таблицами стилей CSS, цифровыми изображениями и т.д., в интерактивное изображение для её представления пользователю.

Первые браузеры были монолитными и не имели отдельного веб-движка. Выделение веб-движка в отдельный модуль не только дало толчок развитию новых Интернет-обозревателей, но также сделало возможным интеграцию в самые различные приложения функциональности отображения композитных HTML-документов. Так, к примеру, большинство современных почтовых клиентов используют те или иные браузерные движки для отображения почтовых сообщений, использующих HTML-форматирование; они лежат в основе различных систем автоматической справки, — таких, как, например, Qt Assistant, который упоминался ранее в настоящем курсе. Одно из популярных направлений использования браузерных движков – это создание виджетов рабочего стола, которые могут являться, по сути, Интернет-приложениями, но при этом отображаться на рабочем столе, а не в браузере.

8.2.2. История создания браузерного движка WebKit

WebKit – это свободный браузерный движок для отображения веб-страниц, изначально разработанный компанией Apple Inc. на основе кода библиотеки веб-рендеринга KHTML и движка JavaScript KJS, являющихся частью проекта KDE и основой браузера Konqueror. Разработчики Apple приступили к портированию этих библиотек на Mac OS X примерно в начале 2002 года в рамках проекта по созданию браузера Safari. Год спустя, в январе 2003, о разработке было объявлено официально, её код был выпущен в свободный доступ, а разработчикам KHTML было предложено сотрудничать с разработчиками WebKit. В 2003 году WebKit вошел в Mac OS X v10.3 как основа браузера Safari .

В 2005 году все компоненты WebKit были выпущены под открытыми лицензиями (LGPL и BSD-типа), которые подразумевают, что любой из компонентов кода может быть переиспользован как в некоммерческих, так и в коммерческих целях, с одним лишь условием, что библиотеки или их производные должны быть опубликованы с открытым исходным кодом на условиях этих лицензий.

На данный момент движок WebKit осуществляет наиболее полную поддержку HTML в соответствии с рекомендациями W3C. Начиная с 2007 WebKit поддерживает HTML5 . Для

современного браузера очень важным критерием работы является эффективность JS-движка, так как веб-приложения используют много js-скриптов и их функциональность зачастую напрямую зависит от скорости выполнения этих скриптов. Поэтому в 2008 начата работа над JS-движком SquirellFish .

Браузерный движок WebKit был выбран в качестве основы для таких браузеров как Chrome (так как разработка Chrome началась раньше, чем была анонсирована разработка JS-движка SquirellFish, для него был написан свой JS-движок V8), Safari (в том числе и в iPhone), браузера для ОС Android и браузера для S60 (Symbian).

8.2.3. QtWebkit

Как уже упоминалось выше, изначально WebKit разрабатывался в рамках проекта KDE и имел много зависимостей от фреймворка Qt. В процессе портирования эти зависимости были устранены, и WebKit в своем текущем состоянии от фреймворка Qt не зависит. Для пользователей Qt была разработана возможность использования движка WebKit через Qt в виде обертки QtWebKit, которая по сути является адаптацией интерфейса движка WebKit на Qt. Начиная с версии Qt 4.4 QtWebKit является частью фреймворка. QtWebKit поддерживает платформы Linux (в том числе MeeGo), Windows, Mac OS X и Symbian.

QtWebKit состоит из следующих базовых компонент:

- `QWebView` — базовый элемент, widget, который умеет отображать веб-страницы. Рассмотрим небольшой примериспользования QWebView, в котором мы передаем объекту типа `QWebView` URL страницы и просматриваем ее:

```
QWebView *view = new QWebView(this);
view->load(QUrl("http://qt.nokia.com/"));
view->show();
```

- `QWebPage` — страница, отображаемая в QWebView
- `QWebFrame` — фрейм. Страница может состоять из нескольких фреймов.
- Взаимодействие с WebKit при этом не ограничивается отображением страниц. Можно получить структуру страницы, разобрать ее и получить доступ к элементу, представленному классом `QWebElement`, элементом DOM-дерева, представляющего `QWebFrame` .

QtWebKit позволяет разрабатывать различные приложения, использующие основную функциональность браузера, под фреймворком Qt. Для знакомства с QtWebKit можно обратиться к стандартным примерам, входящим в поставку фреймворка, — например, создание клиента для GTalk.

8.3. libsocialweb

libsocialweb – это общий back-end для приложений, работающих с социальными сетями . В прошлой лекции мы обсуждали фреймворк telepathy, отвечающий за взаимодействие с системами обмена мгновенными сообщениями, голосовой и видеотелефонии. Идея была такова, что существует множество сервисов, работающих с системой обмена мгновенными сообщениями телефонией, и есть много клиентов, которые используют эти функциональности. При этом существует дополнительный сервис, который, с одной стороны, осуществляет всю работу с различными сервисами, а с другой — предоставляет простой единый интерфейс, при помощи которого пользовательские приложения могут использовать функционал этих сервисов.

Libsocialweb идейно очень похож на сервис telepathy. Он скрывает за собой и унифицирует программные интерфейсы различных сервисов блоггинга. Он поддерживает такие сервисы, как Flickr, Last.fm, Twitter, Vimeo, Facebook (при помощи стороннего plugin'a) и предоставляет приложениям простой интерфейс, который позволяет им отправлять сообщения (каковыми могут являться также изображения и видеоролики) в этих сервисах, читать ленту сообщений друзей, выполнять поиск. Таким образом, если вы намереваетесь создать приложение, которое интегрируется с несколькими социальными сетями и сервисами блоггинга, вам более нет необходимости изучать протокол взаимодействия с этими сервисами или искать различные библиотеки для работы с ними; достаточно

лишь освоить несложный программный интерфейс libsocialweb. Как и telepathy, libsocialweb представляет собой программу-демон, взаимодействие с которой осуществляется через шину D-Bus; как и в telepathy, поддержка различных сервисов реализована в отдельных модулях. Поддержка нового сервиса может быть легко добавлена при помощи реализации соответствующего плагина.

В рамках проекта MeeGo была также создана Qt-обёртка для libsocialweb — libsocialweb-qt. В настоящий момент работа над этой библиотекой находится в активной фазе, и её уже в ближайшее время можно будет использовать в Qt-приложениях.

8.4. Определение местоположения в мобильных устройствах

8.4.1. GPS

GPS – историческая справка

Идея создания спутниковой навигации родилась ещё в 50-е годы XX века. В тот момент, когда СССР был запущен первый искусственный спутник Земли, американские учёные во главе с Ричардом Кершнером, наблюдали сигнал, исходящий от советского спутника и обнаружили, что благодаря эффекту Доплера частота принимаемого сигнала увеличивается при приближении спутника и уменьшается при его отдалении. Суть открытия заключалась в том, что если точно знать свои координаты на Земле, то становится возможным измерить положение и скорость спутника, и наоборот, точно зная положение спутника, можно определить собственную скорость и координаты.

Реализована эта идея была через 20 лет. В 1973 году была запущена программа DNSS, позже переименованная в Navstar-GPS, а затем просто в GPS. Первый опытный спутник был выведен США на орбиту 14 июля 1974 г, а последний из всех 24 спутников, необходимых для полного покрытия земной поверхности, был выведен на орбиту в 1993 г., таким образом, GPS встала на вооружение. Стало возможным использовать GPS для точного наведения ракет на неподвижные, а затем и на подвижные объекты в воздухе и на земле.

Первоначально GPS — глобальная система позиционирования, — разрабатывалась как исключительно военный проект. Но после того, как в 1983 году был сбит вторгшийся в воздушное пространство Советского Союза из-за ошибки в навигации самолёт Корейских Авиалиний с 269 пассажирами на борту, президент США Рональд Рейган разрешил частичное использование системы навигации для гражданских целей. Во избежание применения системы для военных нужд точность незашифрованного сигнала, принимаемого гражданскими приёмниками, намеренно искажалась, что снижало точность определения местоположения. По основному, зашифрованному каналу, предназначенному для использования вооруженными силами США и дружественных им стран, передавался максимально точный сигнал.

В 2000 г. властями США было принято решение прекратить искажение незашифрованного, "гражданского" сигнала. Точность определения координат в современных приёмниках зависит теперь, таким образом, лишь от метеоусловий и от эффективности используемых приемником алгоритмов.

GPS – принцип работы

Принцип работы спутниковых систем навигации основан на измерении расстояния от антенны на объекте (координаты которого необходимо получить) до спутников, положение которых известно с большой точностью. Таблица положений всех спутников называется альманахом, которым должен располагать любой спутниковый приёмник до начала измерений. Обычно приёмник сохраняет альманах в памяти со времени последнего выключения и если он не устарел — мгновенно использует его. В случае же, если альманах устарел, приёмник получает его в сигнале со спутника, который содержит как данные альманаха, так и эфемериду — точную орбиту самого спутника. Таким образом, зная расстояния до нескольких спутников системы, с помощью обычных геометрических построений, на основе альманаха, можно вычислить положение объекта в пространстве (на Рис. 8.4.1 показано, что местоположение объекта определяется как пересечение нескольких сфер, радиусы которых равны расстоянию от объекта до спутников, найденное по разнице во времени, а центры совпадает с местоположением спутников).

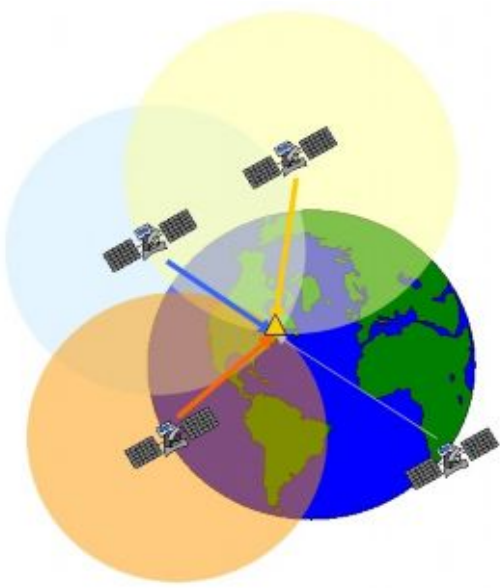


Рис.8.4.1.

Метод измерения расстояния от спутника до антенны приёмника основан на определённости скорости распространения радиоволн. Для осуществления возможности измерения времени распространения радиосигнала каждый спутник навигационной системы в составе своего сигнала излучает сигналы точного времени, используя для его определения точные атомные часы. При работе GPS-приёмника его часы синхронизируются с системным временем и при дальнейшем приёме сигналов вычисляется задержка между временем излучения, содержащимся в самом сигнале, и временем приёма сигнала. Располагая этой информацией, навигационный приёмник вычисляет координаты антенны. Дополнительно накапливая и обрабатывая данные за определённый промежуток времени, становится возможным вычислить такие параметры движения, как скорость (текущую, максимальную, среднюю), пройденный путь и т. д.

В реальности работа системы происходит значительно сложнее. Есть много проблем с получением GPS-координат, требующих дополнительных решений. Прежде всего это связано с тем, что спутники расположены далеко от объекта и мощность GPS-сигнала в среднем очень мала. В общем случае GPS-приемник может использоваться только на открытой местности. Также к проблемам можно отнести большую продолжительность “холодного” старта приемника. Холодным стартом называется процесс сканирования всего частотного диапазона GPS для определения приемником своего местоположения при включении, когда данные альманаха и эфемерид видимых спутников приёмнику неизвестны или устарели. "Холодный" старт для большинства устройств занимает несколько минут, а в некоторых случаях может занимать 10 минут и более. Точная продолжительность зависит от ряда факторов, включая количество видимых спутников и алгоритм поиска, реализованный в данной модели. Следует отметить, что описанная далее технология A-GPS частично решает проблему долгого "холодного" старта.

Технология A-GPS (Assisted GPS) применяется, в первую очередь, для сокращения времени между включением GPS-устройства и определением им текущих координат. В основе этой технологии лежит простая идея: для получения данных альманаха и эфемериды спутников использовать не в крайней степени медленный и ненадёжный радиоканал, а какой-нибудь другой. Обычно при использовании A-GPS эти данные принимаются по каналу мобильного Интернета. Разумеется, для этого в смартфоне или коммуникаторе должно быть настроено Интернет соединение.

Ещё одна существенная проблема, связанная с использованием GPS для определения координат, заключается в том, что большинство GPS-модулей имеют достаточно высокое энергопотребление при работе. И если некоторые специализированные туристические приёмники могут работать непрерывно

в течении суток и более, то для большинства смартфонов время работы с включённым GPS не превышает нескольких часов.

8.4.2. Определение местоположения по Cell ID

Кроме GPS существуют и более простые методы определения местоположения объекта. К ним относится метод определения местоположения по Cell ID. Cell ID – это уникальный идентификатор базовой станции сотовой сети. Как правило, в городских условиях мобильное устройство всегда находится в зоне видимости нескольких базовых станций. Теоретически, метод определения местоположения объекта с использованием данных сотовой сети, мог бы выглядеть следующим образом: мобильное устройство измеряет время прохождения сигнала до него от базовой станции. Первое измерение устройство выполняет для сигнала, посланного к базовой станции, обслуживающей радиосоту, в которой находится данный телефон. Затем аналогичные измерения выполняются для сигналов двух соседних базовых станций. Поскольку координаты базовых станциями известны, с помощью классической триангуляции сравнением трех измеренных временных интервалов можно определить местоположение мобильного терминала.

Но этот метод безупречно работает только в теории. На практике существует существенные сложности для получения местоположения объекта таким образом. Прежде всего это связано с тем, что в большинстве пользовательских устройств нельзя получить никакой служебной информации о сотовой сети, кроме идентификатора текущей соты. Поэтому на деле устройству чаще всего приходится оценивать свое местоположение лишь по идентификатору базовой станции, к которой он в данный момент подключен. Однако поскольку в городе базовые станции обычно расположены весьма близко друг к другу, на практике устройству для грубой оценки своего местоположения бывает достаточно и этой информации. При этом точность измерения колеблется от сотни метров до нескольких километров.

Существуют открытые и закрытые базы данных координат базовых станций. Устройство может по Интернету передать идентификатор текущей базовой станции одному из таких сервисов и запросить координаты этой станции. Такие приложения как Google Maps для мобильных устройств используют в первую очередь именно этот метод определения местоположения объекта. Метод Cell ID лишен двух существенных недостатков GPS: при его использовании координаты определяются практически моментально, а общее энергопотребление почти не возрастает. Среди недостатков этого метода следует упомянуть то, что он активно использует мобильный Интернет, стоимость которого во многих странах ещё достаточно высока и то, что помимо тех неточностей в определении местоположения, которые изначально заложены в данный метод, и которые были описаны выше, он подвержен ещё более существенным неточностям, вызванным чаще всего неверной привязкой идентификатора базовой станции или коллизией идентификаторов в базе данных. Так, в некоторых случаях этот метод ошибочно определяет, что вы находитесь в другой стране, возможно, в тысячах километров от вашего реального местоположения.

8.4.3. WLAN BSSID

Для определения местоположения объектов также могут использоваться идентификаторы точек доступа Wi-Fi. Этот метод основан на том, что в современных городах есть большое количество точек доступа сети Wi-Fi, которые хоть и не так фиксированы как базовые станции, но обычно тоже неподвижны. Для определения местоположения по WLAN BSSID (идентификатору точки доступа, который представляет из себя число наподобие MAC-адреса) строятся базы данных, которые содержат привязки идентификаторов сетей к координатам. Чаще всего эти базы строятся на основе данных полученных от самих пользователей, возможно в автоматическом режиме: для географической привязки достаточно того, чтоб в устройстве, на котором запущено приложение для сбора данных, были одновременно включены Wi-Fi- и GPS-модули.

Поскольку зона приёма для обычно Wi-Fi точки не превышает 200-300 метров, точность определения местоположения по идентификаторам точек доступа Wi-Fi весьма высока. Однако это

метод обладает и существенным недостатком, состоящим в том, что площадь покрытия Wi-Fi сетей, на которой возможно использование такого метода определения координат, на порядок ниже площади покрытия сетями сотовой связи.

8.4.4. IP location

IP location можно отнести к самым неточным методам определения местоположения. Он использует географическую привязку IP адресов, получаемых пользователем при подключении к Интернету. Чаще всего этот подход не способен дать более точной географической привязки, чем название населенного пункта, однако для некоторых приложений такой информации, бывает вполне достаточно для работы. Так, если вы разрабатываете приложение, которое отображает погоду в текущем местоположении пользователя, то сведений о городе, в котором находится пользователь, будет вполне достаточно.

8.5. Определение местоположения в MeeGo

Для различных сервисов определения местоположения в MeeGo существует удобная обертка GeoClue . GeoClue представляет из себя общий сервер для сервисов, работающих с

местоположением (подобно libsocialweb и telepathy). Действует GeoClue точно по такому же принципу как и telepathy: существуют сервисы, которые работают с местоположением, — например, определяющие, текущие координаты одним из описанных выше способов, или преобразующие координаты в названия и адреса и наоборот (это действие обычно называется Geocoding). Geoclue берет на себя взаимодействие с этими сервисами, предоставляя пользовательским приложениям простой D-Bus интерфейс. Любой новый сервис, выполняющий ту или иную задачу с определением местоположения, можно реализовать как плагин Geoclue, тем самым расширив его функциональность.

Geoclue поддерживает следующие методы определения местоположения:

1. GPS и A-GPS: информация о местоположении получается от GPS-приемника
2. GSM: информация о местоположении из сети сотовой связи
3. Plazes: информация о местоположение определяется по видимым точкам доступа Wi-Fi
4. Hostip : информация о местоположении получается по IP-адресу

8.6. QtMobility

В наших лекциях мы уже не раз упоминали о библиотеке QtMobility, которая должна стать основой для разработки приложений для мобильных устройств. Несмотря на то, что в QtMobility существует API для сервисов определения местоположения, в настоящий момент среди провайдеров местоположения там представлен лишь GPS-приёмник. В ближайшее время планируется работа по существенному расширению поддерживаемых провайдеров.

8.7. Лабораторная работа № 6 «MeeGo сервисы Internet&Location»

8.7.1. Цель лабораторной работы

Научиться использованию в своих приложениях сервисов Интернета и определения местоположения.

8.7.2. Введение

План

- Подготовка
  - установка необходимых программных пакетов
  - сборка и установка geoclue, конфигурация динамического линковщика

- установка веб-браузера chrome либо веб-бразуера firefox с расширением Firebug
- Приложение MapView
  - QtWebKit — напоминание.
  - Сборка и запуск приложения MapView. Знакомство с его интерфейсом и реализацией.
  - Добавление поддержки сервиса openstreetmap. Реализация пост-обработки для модификации веб-интерфейса.
  - Реализация получения текущих координат от geoclue
  - Загрузка карты по названию/адресу объекта (geocoding) при помощи geoclue

Необходимые знания и навыки

- Знакомство с материалом лаб. работ №№ 2, 3
- Базовое знание языка программирования C++
- Базовое знакомство с фреймворком Qt и механизмом сигналов и слотов (см. лаб. работу №3)
- Базовое знакомство с основными служебными программами Linux (ls, rm, mkdir и т. п.) и принципами работы систем управления пакетами
- Желательно базовое знакомство с языком разметки HTML и таблицами стилей CSS

Необходимые программные и аппаратные средства

- ПК под ОС Linux (поддерживаются дистрибутивы Fedora 13, Ubuntu 10.04, openSUSE 11.3)
- Подключение к Интернету

8.7.3. Инструкция по выполнению лабораторной работы

8.7.3.1. Подготовка

Установка необходимых пакетов мими

Указания даны для дистрибутива Ubuntu 10.04

- Установить (в дополнение к пакетам, установленным в предыдущих работах) следующие пакеты при помощи команды

```
apt-get install
```

  - qt4-sdk

Сборка и установка geoclue

Установим версию библиотеки 0.11.1

- Ссылка для скачивания пакета исходных файлов: http://folks.o-hand.com/jku/geoclue-releases/geoclue-0.11.1.tar.gz
- Распаковываем архив (см. лаб.работу №2)
- Запускаем скрипт конфигурации:

```
./configure
```

- Запускаем сборку:

```
make all
```

- Устанавливаем библиотеку:

```
sudo make install
```

Конфигурация динамического линковщика

Для того, чтоб отделить устанавливаемую «рабочую» версию пакета от пакетов, установленных в системе постоянно, заголовочные файлы и объектный файл (*.so) устанавливаются, соответственно, в каталоги /usr/local/includes и /usr/local/lib, а не в каталоги /usr/includes и /usr/lib, которые обычно используются для этих целей. Это, однако, означает, что при сборке будет необходимо указать правильный префикс каталога, по которому следует искать заголовочные файлы и файл библиотеки. Необходимо также убедиться, что правильный файл *.so находится в кеше динамического линковщика, который отвечает за подключение прилинкованных динамически библиотек во время загрузки приложения.

- Проверьте, находится ли установленная библиотека в кеше динамического линковщика.

```
ldconfig -p | grep geoclue
```

- Если команда не дала какого-либо вывода, добавляем строку "/usr/local/lib" в файл /etc/ld.so.conf.d/usr-local-lib.conf
- Выполняем

```
sudo ldconfig
```

для обновления кеша

Подготовка инструментов для анализа web-страниц

Приложение, которое будет создано в ходе настоящей работы, будет модифицировать веб-страницы картографических сервисов для более удобного представления их пользователю. Для этого нам понадобится изучить структуру веб-страницы и определить, где и как именно определяются в html-коде различные элементы интерфейса. Для анализа веб-страниц удобно использовать один из следующих инструментов:

- браузер chrome (имеет функцию Developer Toolbar)
- браузер Firefox + расширение Firebug

Давайте установим один из этих инструментов:

- Установка chrome:
  - Перейдите на http://www.google.com/chrome и следуйте указаниям
- Установка Firefox + Firebug
  - Перейдите на http://www.mozilla.com/firefox/ и следуйте указаниям
  - Запустив Firefox, перейдите на https://addons.mozilla.org/firefox/addon/firebug/
  - Нажмите на кнопку «Add to Firefox» и следуйте указаниям

8.7.3.2. Приложение MapView

QtWebKit – напоминание

QtWebKit — это Qt-обёртка для браузерного движка WebKit, позволяющая не только встраивать в Qt-приложение браузерный компонент и представлять различные HTML-документы, но также взаимодействовать с DOM-деревом, представляющим страницу и исполнять в его контексте произвольный javascript-код. Приведём вновь основные классы модуля QtWebKit:

- QWebView — виджет для просмотра и редактирования веб-документов.
- QWebPage — веб-документ (веб-страница). Доступ через QWebPage::page().
- QWebFrame — фрейм (может быть несколько на страницу).
- QWebElement — элемент DOM-дерева, составляющего фрейм.

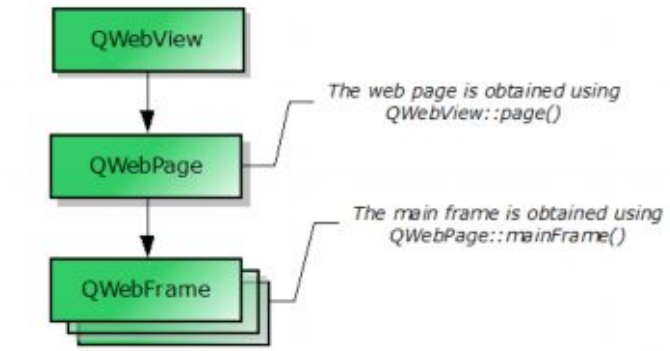


Рис. 8.7.1.

Сборка приложения MapView

Рассматриваемое в настоящей работе приложение MapView позволяет просматривать карты заданной местности параллельно в нескольких online картографических сервисах. Давайте соберём и запустим его:

- Распакуйте архив с исходным кодом приложения:

```
tar xzf mapview.tar.gz
```

- Перейдите в созданный каталог

```
cd mapview
```

- Сгенерируйте Makefile из файла *.pro

```
qmake
```

- Соберите приложение

```
make all
```

- Запустите приложение

```
./mapsview
```

Интерфейс MapView

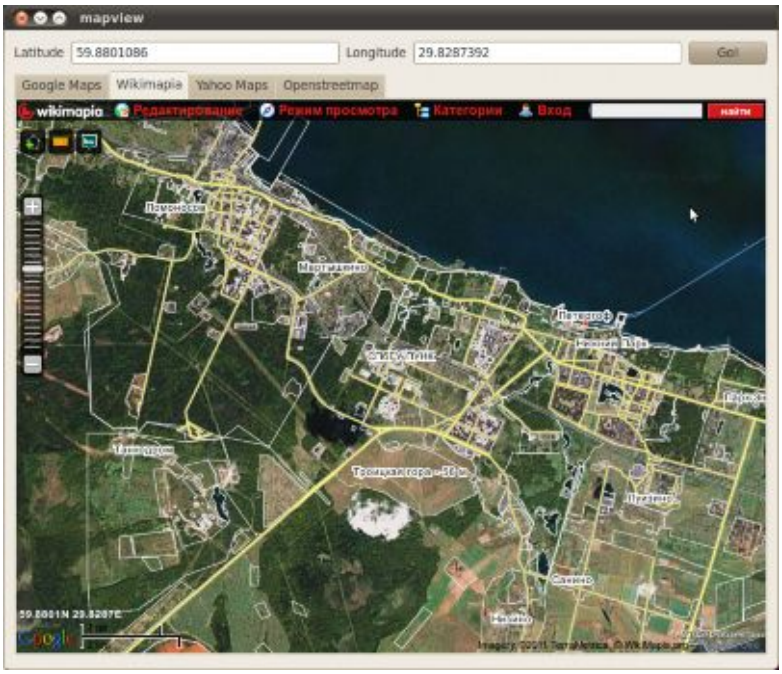


Рис. 8.7.2.

- В текстовые поля в верхней части окна вводятся координаты
- При нажатии на кнопку «Go!» веб-страницы Google Maps, Wikimapia, Yahoo Maps загружаются в виджеты QWebView, размещенные на соответствующих вкладках
- Поддержка сервиса Openstreetmap будет добавлена в ходе выполнения задания

Задание: поддержка сервиса Openstreetmap

- Перейдите в браузере на http://www.openstreetmap.org
- Получите ссылку на карту, нажав на «Permalink» в правом нижнем углу карты и изучите структуру ссылки (где в ней задаются координаты)
- Добавьте загрузку страницы с картой OSM в функцию `loadOSMMap()`, используя в качестве образцов функции `loadWikimapiaMap()` и `loadYahooMap()`.

Работа с DOM-представлением веб-документа

- Цель: убрать из представления лишние элементы и оставить только карту, аналогично тому, как это делается в `loadYahooMap()`.
- Способ: найти узлы DOM-дерева, отвечающие за «лишние» элементы и выставить параметр стиля "visibility" в значение "hidden".
  - Изучаем структуру документа при помощи Firebug или Developer Tools для Chrome и записываем значение атрибута "id" «ненужных» узлов

- Создаём слот onOSMLoadFinished() и подключаем к нему сигнал `loadFinished()` виджета mOSMView
- В реализации слота находим узлы с заданным id

```
QWebElement el = mOSMView->page()->mainFrame()->                findFirstElement("#id")
```

- Скрываем их

```
el.setStyleProprty("visibility", "hidden")
```

Как использовать Chrome Developer Tools

Замечание: Мы не будем рассматривать здесь работу с расширением Firebug для браузера Firefox. Для самостоятельного изучения можно обратиться к документации по этому расширению: http://getfirebug.com/wiki/.

- Откройте Developer Tools нажатием Ctrl+Shift+I.
- Выберите вкладку "Elements" на открывшейся панели.
- Нажмите на изображение лупы в левом нижнем углу.

- Наведите курсор на элемент представления страницы, который вы хотите изучить, и щёлкните по нему.
- Обозреватель DOM будет открыт на выбранном элементе.

Работа с geoclue. Определение собственного местоположения

Цель: сделать так чтоб приложение MapView открывалось на текущем местоположении.

- Запустите пример examples/position-example из поставки geoclue, используя провайдер Hostip.
- Hostip является открытым проектом по созданию БД IP-адресов с привязкой к их местоположению. Возможно, ваш IP-адрес не будет найден в этой базе. В этом случае, используйте форму на сайте http://hostip.info для добавления вашего IP-адреса в базу.
- Изучите код примера (examples/position-example.c).
- На основании примера реализуйте получение текущих координат с использованием провайдера Hostip при загрузке приложения MapsView.

Работа с geoclue. Geocodding

Цель: добавить в приложение MapView возможность задания местоположения при помощи geocoding (получение координат по названию/адресу объекта).

- Запустите пример examples/geocode-example из поставки geoclue с провайдером Yahoo
- Изучите код примера:
  - examples/geocode-example.c
- Добавьте в интерфейс MapsView поле ввода (QTextEdit) и кнопку (QPushButton) "Geocode".
- Руководствуясь примером geocode-example реализуйте слот loadGeocoded(), который будет получать координаты заданного объекта от провайдера Yahoo и загружать карты.
- В случае, если установить координаты не удалось, следует выдать предупреждение при помощи QMessageBox.

8.8. Выводы

В этой лекции мы рассмотрели очередную важную подсистему ОС MeeGo, которая ответственна за работу с Интернетом и определение местоположения. Мы познакомились с популярным браузерным движком WebKit и узнали о его роли в MeeGo. Мы провели обзор библиотеки libsocialweb, которая отвечает в MeeGo за взаимодействие с социальными сетями и сервисами блоггинга (предоставляет единый интерфейс для приложений, которые работают в социальных сетях). Заключительная часть нашей лекции была посвящена методам определения местоположения и

модулю GeoClue, который предоставляет клиентским приложениям упрощенный интерфейс доступа к сервисам определения местоположения.

8.9. Контрольные вопросы

1) Что такое браузерный движок?
   1. программная компонента, используемая только в браузерах для представления веб-страниц
   2. программная компонента, отвечающая за представление веб-страниц; является основной браузера, но может быть переиспользована другими приложениями
   3. программная компонента, лежащая в основе веб-сервера Apache
   4. CMS (content management system), которую использует компания Intel на своих сайтах
2) Какое основное преимущество дает выделение браузерного движка в отдельный модуль?
   1. улучшение читаемости кода браузера
   2. возможность использовать функциональность веб-движка в других приложениях
   3. улучшение качества тестирования компонента, так как в браузерном движке находится наибольшее количество багов
   4. выделение браузерного движка в отдельный модуль не даёт особых преимуществ. это делается исключительно по лицензионным соображениям.
3) Что такое WebKit?
   1. браузерный движок, разработанный компанией Apple на основе движка KHTML и Javascript- движка KJS
   2. браузерный движок, разработанный компанией Apple на основе движка Gecko
   3. браузерный движок, разработанный компанией Google на основе движка KHTML и Javascript-движка KJS
   4. популярный SDK для разработки web-приложений
4) Что такое SquirellFish?
   1. веб-движок
   2. браузер
   3. встроенный отладчик для JavaScript
   4. фреймворк для обработки сценариев JavaScript
5) Чем была обусловлена необходимость разработки SquirellFish?
   1. в современных сайтах большое количество JavaScript-кода и от качества его выполнения сильно зависят впечатления пользователя
   2. старые вырсии веб-движка WebKit не имели фреймворка для обработки JavaScript
   3. прежний фреймворк JavaScriptCore имел очень много критических багов
   4. возникли проблемы с лицензиями, которые привели к необходимости разработать этот модуль заново
6) Какой из следующих браузеров не основан на WebKit?
   1. Safari (Mac OS X и iOS)
   2. Chromium /Google Chrome
   3. Android web browser
   4. Web Browser for S60 (Symbian)
   5. Firefox
7) Какой компонент фреймворка Qt позволяет использовать WebKit?
   1. модуль QtWebKit
   2. модуль QtWebKit, который, однако, не входит в поставку Qt и не поддерживается официально

3. WebKit был создан на базе KHTML, и поэтому написан целиком с использованием Qt. Поэтому отдельного модуля для поддержки WebKit в Qt нет.
4. В настоящий момент для WebKit не существует интерфейса, который позволил бы использовать его в Qt-приложениях.

8) Какие платформы не поддерживает QtWebKit?
   1. MeeGo
   2. Android
   3. Windows
   4. Mac OS X
   5. Symbian
9) Укажите, что из списка не является классом QtWebKit?
   1. QWebView
   2. QWebPage
   3. QtWebEdit
   4. QWebFrame
   5. QWebElement
10) Для чего была разработана библиотека libsocialweb?
   1. для взаимодействия с сервисами социальных сетей
   2. для взаимодействия с on-line магазинами
   3. для взаимодействия с сервисами обмена мгновенными сообщениями
   4. для социальных исследований: libsocialweb позволяет построить граф друзей в сетях Facebook, MySpace и некоторых других
11) Укажите, какой сервис из списка не поддерживает библиотека libsocialweb?
   1. Flickr
   2. Last.fm
   3. Vkontakte
   4. Vimeo
   5. Facebook
12) Что из указанного является неверным утверждением?
   1. GPS - система глобальной спутниковой навигации
   2. GPS разработана министерством обороны США
   3. гражданские приёмники GPS дают значительно более низкую точность определения координат, чем военные
   4. для определения координат приёмник должен принять сигналы, как минимум, с трёх спутников
13) Какое утверждение наиболее точно описывает принцип действия GPS?
   1. Пользовательское GPS-устройство передаёт сигнал, улавливаемый направленной принимающей антенной спутника. Зная характер земной поверхности в точке, откуда был излучён сигнал, спутник вычисляет местоположение устройства и передаёт его в своём сигнале.
   2. Пользовательское GPS-устройство принимает в сигнале со спутников точные параметры их орбиты и вычисляет удаление от них на основе времени прохода сигнала. Затем, при помощи триангуляции вычисляется местоположение.
   3. В память пользовательского GPS-устройства изначально заложены данные об орбитах спутников GPS. При помощи направленной антенны оно определяет местонахождение спутников на небесной полусфере. Сопоставляя первоначальные данные об орбитах спутников и данные об их расположении на небе, устройство определяет своё местоположение.

4. пользовательское GPS-устройство передаёт на спутник идентификаторы базовых станций сотовой связи, в зоне которых это устройство находится. Спутник располагает обновляемой базой данных базовых станций и их местоположений, при помощи которой и устанавливается точное местонахождение пользовательского устройства и передаётся в обратном сигнале.

14) Что из указанного нельзя отнести к недостаткам GPS-приемника?
1. мощность сигнала со спутника очень мала, в результате чего GPS-приемник, в общем случае, может использоваться только на открытой местности.
2. «холодный» старт приёмника занимает, по крайней мере, несколько минут
3. энергопотребление модуля GPS весьма высоко
4. точность определения координат гражданскими (невоенными) приёмниками очень невысока

15) Применительно к технологии GPS, что называется термином "альманах"?
1. "прошивка" клиентского устройства, позволяющая ему декодировать сигнал со спутников и распространяемая раз в 3 года министерством США среди производителей GPS-микросхем
2. ежедневный сеанс связи между центром управления полётами и спутником системы GPS
3. сеанс связи между спутником и пользовательским GPS-устройством
4. данные об орбитах спутников системы GPS и об их работоспособности
5. сигнал точного времени, передаваемый со спутника

16) Что является причиной длительного "холодного" старта GPS-приемника?
1. необходимость точно "навестись" на каждый из спутников, что делается при помощи направленной антенны
2. вычисления большой вычислительной сложности в совокупности с малой вычислительной мощностью GPS-устройства
3. сравнительно большой объём данных, который требуется получить по медленному и ненадёжному каналу
4. задержка при "холодном старте" — это намеренное ограничение, установленное мин. обороны США для гражданских приёмников, позволяющее исключить их применение в некоторых военных целях

17) Что из перечисленного ниже не входит в сигнал, передаваемый спутником GPS?
1. альманах
2. эфемерида
3. удаление от поверхности земли
4. сигнал точного времени
5. данные о работоспособности спутника

18) Каково минимальное число спутников, сигнал которых должен принять конвенциональный GPS-приёмник для того, чтоб определить свои координаты (широту и долготу)?
1. 2
2. 3
3. 5
4. достаточно сигнала с одного спутника, но координаты будут определены с точностью до 3 км.

19) Благодаря чему ускоряется "холодный" старт GPS-приёмника при использовании технологии A-GPS (Assisted GPS)?
1. благодаря использованию данных, передаваемых по зашифрованному каналу со

спутника
2. благодаря использованию вычислительной мощности центрального процессора устройства в первоначальных вычислениях
3. благодаря использованию более быстрого и стабильного канала для получения альманаха и эфемерид
4. благодаря использованию спутников системы ГЛОНАСС в дополнение к спутникам GPS

20) Какой принцип лежит в основе метода определения местоположения по CellID?
1. абонентское устройство определяет своё местоположение как область вокруг базовой станции, к которому оно в данный момент подключено. Местоположение базовой станции устанавливается по её уникальному идентификатору при помощи обращения к соответствующей базе данных.
2. абонентское устройство определяет своё местоположение как область вокруг базовой станции, к которому оно в данный момент подключено. Местоположение базовой станции закодировано в её сигнале.
3. абонентское устройство определяет своё местоположение при помощи триангуляции, исходя из местоположения базовых станций, в зоне видимости которых оно находится. Местоположение БС определяется по их уникальному идентификатору при помощи обращения к соответствующей базе данных.
4. несколько базовых станций сотовой сети вычисляют время прохода сигнала до абонентского устройства, при помощи триангуляции определяют его местоположение и передают его по GPRS-каналу

Список литературы

1. WebKit

http://webkit.org/
http://trac.webkit.org/wiki/QtWebKit
http://doc.qt.nokia.com/4.7-snapshot/qtwebkit.html

2. libsocialweb

http://git.gnome.org/browse/libsocialweb/

3. wiki проекта GeoClue

http://www.freedesktop.org/wiki/Software/GeoClue

4. Документация по Qt

http://doc.qt.nokia.com/4.7/index.html

или при помощи QtAssistant

5. Документация по html и css

http://www.w3.org

6. Документация по chrome developer tools

http://code.google.com/chrome/devtools/

9. MeeGo API: работа с графикой и интернационализация

MeeGo API: Visual Services. 3D Graphics [OpenGL/GLES], 2D Graphics [Cairo, QPainter], I18n Rendering [Pango, QtText], GTK/Clutter. Примеры использования OpenGl, Cairo и локализации приложения по средствам I18n.

9.1. Введение

В этой лекции мы кратко рассмотрим возможности **MeeGo** по работе с двухмерной и трёхмерной графикой (а именно, **OpenGL ES**, **QPainter** и **SVG**), их ключевые особенности и принципы работы. Кроме того, мы подробно охватим средства интернационализации (**i18n**) и локализации MeeGo приложений.

Стоит отметить, что все эти возможности реализуются с помощью фреймворка **Qt**. Соответственно, несмотря на то, что мы ведём лекцию в контексте MeeGo SDK — обсуждение кардинально не отличается от разработки обычных проектов под Qt. Кроме того в лекции используются примеры из официальной документации MeeGo и Qt [1].

9.2. Двухмерная графика

9.2.1. QPainter, QPaintDevice, QPaintEngine

Система рисования Qt позволяет рисовать на экране и любых устройствах вывода используя один и тот же интерфейс, основанный на трёх классах — **QPainter**, **QPaintDevice**, **QPaintEngine**.

QPainter используется для операций рисования, **QPaintDevice** — абстракция двухмерного пространства, на котором можно рисовать используя **QPainter**, и **QPaintEngine** — это интерфейс с помощью которого можно рисовать на разных типах устройств (см. Рис. 9.2.1). Класс **QPaintEngine** используется внутри **QPainter** и **QPaintDevice** и скрыт от глаз разработчика, только если он не создаёт свою собственную реализацию устройства вывода (наследника **QPaintDevice**).

Рис. 9.2.1.

Вывод графики осуществляется как через стандартные виджеты **QWidgets**, так и через другие классы, как то **QImage**, **QPixmap**, **QGLPixelBuffer**, **QPicture**, **QPrinter** и т. д.

QPainter включает в себя высокооптимизированные функции работы с графикой, которые могут использоваться для рисования практически любой графики, которая может понадобится. Например, функции рисования простых линий и сложных форм, подобных секторам круга и хордам, функции отображения выровненного текста и пиксельных рисунков и т.д. Кроме того, возможно изменение систем координат и различные трансформации.

Типичный способ использования **QPainter** — рисование внутри обработчика сообщения.

Простой пример:

```
void SimpleExampleWidget::paintEvent()
{
   QPainter paint(this);
   paint.setPen(Qt::blue);
   paint.drawText(rect(), Qt::AlignCenter, tr("The Text"));
}
```

Существует некоторые настройки, которые можно менять. Приведём часть из них:

- `font()` — текущие установки шрифта;
- `brush()` — текущие установки кисти; цвет или образец используемый для заливки, например, круги;
- `pen()` — текущие настройки пера; цвет или пунктир используемый для рисования линий и

границ;

- backgroundMode() — является ли непрозрачным или прозрачным, т. е. используется или нет background();
- background() применяется только тогда, когда backgroundMode уставлен в «прозрачен»;
- brushOrigin() — оригинальный узор кисти, обычно оригинальный фон виджета;
- viewport(), window(), matrix() — составляют систему преобразования координат (они будут описаны чуть более подробно далее);
- hasClipping() — указывает, осущесвтляет ли класс обрезание по каким-либо границам. Если обрезание осуществляется, то по региону, указанному в методе clipRegion();
- save() — сохраняет настройки во внутреннем стеке;
- restore() — восстанавливает их.

Класс **QPainter** предназначен для рисования и имеет множество примитивов, например: drawPoint(), drawPoints(), drawLine(), drawRect(), drawRoundRect(), drawEllipse(), drawArc(), drawPie(), drawChord(), drawLineSegments(), drawPolyline(), drawPolygon(), drawConvexPolygon(), drawCubicBezier() и т. д.

Также есть функции для рисования на пиксельных картах/рисунках, а именно drawPixmap(), drawImage() и drawTiledPixmap(). И drawPixmap() и drawImage() приводят к одному результату за исключением того, что drawPixmap() быстрее работает на экране, а drawImage() может быть быстрее на экземплярах класса **QPrinter,** или других устройствах.

Текст рисуется с помощью drawText(). Если необходимо точное расположение, то может быть использован метод boundingRect(), возвращающий координаты, где данный drawText() выполнил бы рисование.

Пара примеров:

```
QRectF rectangle(10.0, 20.0, 80.0, 60.0);
int startAngle = 30 * 16;
int spanAngle = 120 * 16;

QPainter painter(this);
painter.drawPie(rectangle, startAngle, spanAngle);
```

Ещё:

```
static const QPointF points[4] = {
    QPointF(10.0, 80.0),
    QPointF(20.0, 10.0),
    QPointF(80.0, 30.0),
    QPointF(90.0, 70.0)
};

QPainter painter(this);
painter.drawPolygon(points, 4);
```

Системы координат

Обычно **QPainter** работает на системе координат устройства, но **QPainter** хорошо поддерживает различные преобразования систем координат. Например, наиболее часто используемыми являются методы scale(), rotate(), translate() и shear(), все они работают с матрицей, возвращаемой по вызову метода matrix(). Матрицу можно изменить и вручную, после чего изменения необходимо развернуть, вызовом метода setMatrix().

Метод setViewport() устанавливает прямоугольник, на котором работает **QPainter**. Метод setWindow() привязывает систему координат к этому прямоугольнику.

Небольшой фрагмент из примера по рисованию аналоговых часов:

```
void AnalogClock::paintEvent(QPaintEvent *)
 {
```

```
static const QPoint hourHand[3] = {
   QPoint(7, 8),
   QPoint(-7, 8),
   QPoint(0, -40)
};
static const QPoint minuteHand[3] = {
   QPoint(7, 8),
   QPoint(-7, 8),
   QPoint(0, -70)
};

QColor hourColor(127, 0, 127);
QColor minuteColor(0, 127, 127, 191);

int side = qMin(width(), height());
QTime time = QTime::currentTime();

QPainter painter(this);
painter.setRenderHint(QPainter::Antialiasing);
painter.translate(width() / 2, height() / 2);
painter.scale(side / 200.0, side / 200.0);
}
```

Обрезание

Методы класса **QPainter** позволяют обрезать любую операцию рисования по границе прямоугольника, области или векторного пути. Управление текущими границами обрезания может осуществляться с помощью функций `clipRegion()` и `clipPath()`.

После обрезания **QPainter**, устройство рисованию также может обрезать. Например, большинство границ обрезания виджетов вне границ обрезания дочерних виджетов, а большинство границ обрезания принтеров находятся далеко от областей обрезания виджетов около границ бумаги. Эти дополнительные границы обрезания не отражаются в значении, возвращаемом `clipRegion()` или `hasClipping()`.

Рисование сложных фигур

Если необходимо нарисовать сложную фигуру — особенно, если необходимо это делать неоднократно, есть смысл создать экземпляр класса **QPainterPath** и рисовать, используя метод `drawPath()`.

Небольшой пример:

```
QPainterPath path;
            path.addRect(20, 20, 60, 60);
path.moveTo(0, 0);
path.cubicTo(99, 0,  50, 50,  99, 99);
path.cubicTo(0, 99,  50, 50,  0, 0);

QPainter painter(this);
    painter.fillRect(0, 0, 100, 100, Qt::white);
    painter.setPen(QPen(QColor(79, 106, 25), 1, Qt::SolidLine,
                      Qt::FlatCap, Qt::MiterJoin));
    painter.setBrush(QColor(122, 163, 39));

    painter.drawPath(path);
```

9.2.2. SVG

По информации википедии SVG (от англ. Scalable Vector Graphics — масштабируемая векторная графика) — язык разметки масштабируемой векторной графики, созданный Консорциумом Всемирной паутины (W3C) и входящий в подмножество расширяемого языка разметки XML, предназначен для описания двумерной векторной и смешанной векторно/растровой графики в формате XML. Поддерживает как неподвижную, так анимированную и интерактивную графику — или, в иных терминах, декларативную и скриптовую. Это открытый стандарт, является рекомендацией консорциума W3C, — организации, разработавшей такие стандарты, как HTML и XHTML. Разрабатывается с 1999 года, в 2001 году вышла 1.1 версия, которая остается актуальной до сегодняшнего дня, в активной разработке версия 1.2. В основу SVG легли языки разметки VML и PGML.

Для отображения SVG графики в Qt имеется модуль **QtSvg**. Соответственно, для того чтобы использовать возможности SVG необходимо добавить в главный хидер директиву

```
#include <QtSvg>
```

и в проектный файл строку

```
QT += svg
```

Основные классы MeeGo (и Qt) для работы с SVG:

- **QSvgGenerator —** класс, осуществляющий рисование;
- **QSvgRenderer —** отображение содержимого SVG файлов;
- **QSvgWidget —** соответствующий виджет, через который отображаются SVG данные.

QSvgWidget и QSvgRenderer

Класс **QSvgWidget** — это виджет, который используется для отображения содержимого SVG файлов. В связке с классом **QSvgRenderer** он используется во многом так же, как и **QLabel —** для отображения текста и растровых изображений. При этом используя процесс рендеринга классом **QSvgRenderer** можно получить чуть больше контроля (например, вывод через классы **QImage** и **QGLWidget**).

Эквивалентные примеры конструирования экземпляра **QSvgWidget**:

```
widget = QSvgWidget(“filename.svg”);
```

или,

```
widget = QSvgWidget();
widget.load(“filename.svg”);
```

или,

```
widget = QSvgWidget(qbyte);
```

где `qbyte` — экземпляр класса **QByteArray**, содержащий сериализованное представление XML SVG файла.

Заметим также, что **QSvgRenderer** может использоваться для рисования SVG графики на любых экземплярах класса **QPainter**. Используя его, SVG графика может быть выведена на любом «устройстве» рисования (например, на экземплярах **QWidget**, **QImage** и **QGLWidget**).

Класс QSvgGenerator

Класс **QSvgGenerator** является устройством для рисования SVG графики. Он спроектирован как «устройство записи», возвращающее данные в специфическом формате.

Для того чтобы получить SVG файл сперва необходимо сконфигурировать выход, путём указания свойства `fileName`, или `outputDevice`. Например:

```
QSvgGenerator generator;
generator.setFileName(path);
generator.setSize(QSize(200, 200));
generator.setViewBox(QRect(0, 0, 200, 200));
generator.setTitle(tr("SVG Generator Example   Drawing"));
generator.setDescription(
tr("An SVG drawing created by the SVG Generator "
```

```
                "Example provided with Qt."));
```

Далее рисование производится через обычный экземпляр класса **QPainter**, точно также как и для других подклассов **QPaintDevice**.

Например:

```
QPainter painter;
painter.begin(&generator);

painter.fillRect(QRect(0, 0, 200, 200), Qt::darkBlue);
painter.translate(145, 10);
painter.setBrush(Qt::white);
painter.drawPath(moon);
painter.translate(-145, -10);

painter.end();
```

9.3. OpenGL ES

OpenGL ES (OpenGL for Embedded Systems — OpenGL для встраиваемых систем) — подмножество графического интерфейса OpenGL, разработанное специально для встраиваемых систем — мобильных телефонов, карманных компьютеров, игровых консолей. OpenGL ES определяется и продвигается консорциумом Khronos Group, в который входят производители программного и аппаратного обеспечения, заинтересованные в открытом API для графики и мультимедиа.

В связи с тем, что он предназначен для использования встроенными системами, он имеет меньший и более строгий API. В MeeGo SDK, а точнее в Qt, есть библиотеки для работы как в стандарте OpenGL ES 1.1, так и 2.0, причём версия OpenGL 1.1 определена по спецификации 1.5 и обратно совместима с версией 1.0.

Основные отличия двух версий состоят в том, что OpenGL ES 1.x предназначен для платформ с фиксированными аппаратными функциями, а его преемник, OpenGL ES 2.x —для программируемых аппаратных средств. В связи с этим эти два формата не совместимы друг с другом. Например, в OpenGL ES 1.x существует ограниченный набор основных вариантов для рисования и освещение объектов, в OpenGL 2.x имеет значительно более короткий графический конвейер. Вместо того чтобы использовать функции преобразования и графический конвеер с фиксированными функциями, 2.x использует специальный шейдерный язык — OpenGL ES Shading Language (GLSL ES). Вместо использования предопределенных функций, программист пишет маленькие шейдерные программы, «рассказывающие» аппаратной платформе подробно, как визуализировать каждый объект. Это означает, что для вывода на экран даже простейшего треугольника требуется глубокое знакомство с основами компьютерной графики, включая шейдеры.

Решение о том, какой формат использовать в своём приложении остаётся на плечах разработчика. Однако стоит заметить, что формат OpenGL ES 2.0 всё же пришёл на смену 1.x.

В связи с тем, что интерфейсы OpenGL ES полностью стандартизированы и реализующие их в Qt классы полностью следуют букве стандартов и по его использованию существует множество учебников, мы не будем рассматривать работу с OpenGL ES.

9.4. Средства интернационализации (I18N)

«Интернационализа́ция — технологические приёмы разработки, упрощающие адаптацию продукта (такого как программное или аппаратное обеспечение) к языковым и культурным особенностям региона (регионов), отличного от того, в котором разрабатывался продукт.» *(ru.wikipedia.org/wiki/Интернационализация).*

В английском языке для слова “internationalization” принято сокращение «i18n». При этом число 18 означает количество пропущенных между «i» и «n» букв.

Интернационализация — это адаптация продукта для потенциального использования практически в любом месте. Интернационализация производится на начальных этапах разработки, причём в некоторых случаях она проста — например, требуется чтобы приложение было возможно использовать на Украине, а в некоторых может потребовать немного больше, чем просто некоторые исправления орфографии. Например, чтобы сделать приложение написанное в России полезным для японских пользователей, или корейские приложения для немецких, будет требоваться, чтобы программное обеспечение не только отображало информацию на разных языках, но и использовало различные методы ввода, кодировка символов, соглашения о представлении услуг, методы отображения дат и т. д.

В MeeGo SDK есть все инструменты для того, чтобы произвести интернационализацию продукта как можно безболезненнее. В MeeGo SDK используются средства предоставляемые фреймворком Qt — они включают в себя все виджеты ввода данных и методы отрисовки текста. Например, встроенный модуль отображения шрифта способен правильно и красиво отобразить текст, одновременно содержащий символы из различных систем письменности. Кроме того, в Qt предлагается встроенная поддержка практически всех используемых языков (в частности, все восточно-азиатские, западные, арабские, кириллические и т. д), а также всех сценариев Unicode 5.1, не требующих специальной обработки.

Во многих системах письменности есть свои особенности, например:

- **Специальное поведение при разрыве строки.** Некоторые из азиатских языков пишутся без пробелов между словами. Разрыв строки может происходить либо после каждого символа (с некоторыми исключениями), как в китайском, японском и корейском языках, или после логические границы слова, как в Таиланде.
- **Двунаправленные письменные формы.** В Арабском и Иврите пишутся справа налево, за исключением номеров и встроенного английского текста (который пишется слева направо). Точное поведение определяется в 9-м технмическом приложении Unicode.
- **Номера интервалов или диакритических знаков (акцентов или умляутов в европейских языках).** Некоторые языки, такие, как вьетнамские широко используют эти знаки, и некоторые символы могут иметь более одной отметки для уточнения произношения.
- **Лигатуры.** В особых контекстах, некоторые пары символов необходимо заменять комбинированным символом формирование лигатуры.

Поддержка различных систем письменности в большинстве случаев прозрачна для программиста и полностью инкапсулируется внутри модуля вывода текста Qt. Это, в свою очередь означает, что программисту нет никакой необходимости помнить об используемой системе письменности, за исключением нескольких моментов:

- `QPainter::DrawText (int X, int Y, const QString &str)` всегда рисует строку с ее левого края в положение, определенном параметрами X, Y. Для большинства локалей выведется строка, выровненная по левому краю. На арабском языке и иврите в приложениях, как правило, необходимо выравнивать строку по правому краю, так что для этих языков необходимо передавать в функцию `DrawText ()` другие координаты.
- Когда вы пишете собственные элементы управления вводом текста, используйте `QTextLayout`. В некоторых языках (например, арабский язык или языки из Индии), ширина и форма знака меняется в зависимости от окружающих символов, которые `QTextLayout` принимает во внимание. При написании таких элементов управления настоятельно рекомендуется наследоваться от `QLineEdit`, или `QTextEdit`, а также учитывать различные контексты использования.

Общая схема, используемая при интернационализации проекта:

- все литерные тексты необходимые для перевода маркируются прямо в коде по средствам вызова функции `tr()`, или макроса `QT_TR_NOOP()`;
- выполняется трансляция утилитой `lupdate`, которая «выдирает» маркированные тексты;
- тексты переводятся вручную, с использованием утилиты Qt `Linguist`.

9.4.1. QString, tr*(), QT_TR*_NOOP

Чтобы обеспечить интернационализацию, для каждого *литерного текста (текста в кавычках)* необходимо использовать функцию *[*`QCoreApplication::`*]*`tr()` (или её синоним — функцию `translate()`). Она одновременно является маркером для утилиты `lupdate` (о которой подробнее речь пойдёт дальше) и статической функцией C++, осуществляющей перевод.

Например, предполагая, что **LoginWidget** — подкласс **QWidget**:

```
LoginWidget::LoginWidget()
{
    QLabel *label = new QLabel(tr("Password:"));
    ...
}
```

Этот пример подходит для 99% выводимых пользователю строк.

Внимание! *Необходимо быть очень осторожным в использовании функции* `tr()`. Например, такой пример будет работать:

```
QPushButton *ok = new QPushButton( tr("OK"),this);
```

А этот — нет:

```
QCString str = "OK";
QPushButton *ok = new QPushButton( tr(str),this);
```

Точнее говоря, в результате работы второго примера выведется непереведённая строка «OK». Это произойдёт в связи с тем, что утилита для интернационализации `lupdate` «не увидит» связанное значение переменной.

В литерный текст, передающийся в функцию `tr()`, как и в `QString` можно подставлять аргументы, например:

```
tr( QString("Cannot open %1").arg(fileName) )
```

и

```
tr( "Cannot open %1" ).arg( fileName )
```

эквивалентны. Однако, следует пользоваться вторым вариантом, так как первый осложняет работу транслятора `lupdate`.

В отличие от функций `tr()`, макрос `QT_TR_NOOP()` является лишь маркером и не осуществляет перевод заключённого в него текста. Пример:

```
const char *strings[2] = {
     QT_TR_NOOP("OK"),
     QT_TR_NOOP("Cancel")
};
for ( int i = 0; i < 2; i++ ) {
     QButton *but = new QPushButton(strings[i],this);
}
```

В этом примере, возможно неожиданно для разработчика, отобразится кнопка с непереведённым текстом. Для того чтобы вывести переведённый текст, необходимо было писать:

```
QButton *but = new QPushButton( tr(strings[i]),this);
```

9.4.2. Трансляция приложения

Трансляция MeeGo приложений осуществляется в три шага:

1. Необходимо запустить утилиту `lupdate`, которая возьмёт переводимые тексты из исходного C++ кода и поместит их в **.TS** файл (обычно — по одному на язык). Утилита распознаёт вызовы функций `tr()` и макрос `QT_TR*_NOOP()`.
2. Отредактировать получившийся TS файл для того, чтобы перевести текст (вручную (TS файл содержит правильный XML), либо используя утилиту `Qt Linguist`).
3. Вызвать утилиту `lrelease` для того чтобы получить легковесный файл с сообщениями (.QM

файл) из .TS файла, пригодный только для конечного использования. TS файлы можно рассматривать как исходные файлы, а QM, как объектные.

Обычно для каждого релиза приложения данные шаги необходимо повторить. Утилита `lupdate` принимает это во внимание и позволяет использовать заново уже имеющиеся переводы.

Заметим также, что перед тем как запустить утилиту `lupdate` необходимо верным образом подготовить проектный файл (.pro). Пример подобного проектного файла:

```
HEADERS = funnydialog.h \
                      wackywidget.h
SOURCES = funnydialog.cpp \
                  main.cpp \
                  wackywidget.cpp
FORMS = fancybox.ui
TRANSLATIONS = superapp_dk.ts \
                  superapp_fi.ts \
                  superapp_no.ts \
                  superapp_se.ts
```

В этом примере используются Датский, Финский, Норвежский и Шведский языки (это автоматически распознаётся по окончаниям _dk, _fi, _no, _se). В проекте необходимо вызвать методы `QTranslator::load()` с указанием файла перевода и подходящий файл перевода для языка пользователя, а затем установить его используя функцию `QCoreApplication::installTranslator()`.

Типичный пример функции `main` проекта:

```
int main(int argc, char *argv[])
 {
     QApplication app(argc, argv);

     QTranslator qtTranslator;
     qtTranslator.load("qt_" + QLocale::system().name(),
             QLibraryInfo::location(QLibraryInfo::TranslationsPath));
     app.installTranslator(&qtTranslator);

     QTranslator myappTranslator;
     myappTranslator.load("myapp_" + QLocale::system().name());
     app.installTranslator(&myappTranslator);

     ...
     return app.exec();
 }
```

9.4.3. Кодировки

В Qt и MeeGo класс **QString** внутри использует кодировку Unicode 5.1 — т.е. каждый язык мира может быть обработан абсолютно прозрачным образом. Однако, иногда может понадобиться использование различных кодировок. Например, обработка текста в кодировке KOI8-R и её перевод в UTF-8. Для этого применяется специальный класс — **QTextCodec**. С его помощью перевод из кириллической кодировки cp1251 в UTF-8 может быть осуществлён как:

```
QByteArray encodedString = ...; // some cp1251 encoded text

QTextCodec *codec = QTextCodec::codecForName("cp1251");
QString string = codec->toUnicode(encodedString);
```

В идеале, для переносимости необходимо везде использовать кодировку UTF-8 и UTF-16. Наиболее необходимая кодировка для поддержки (в контексте локали) выдаётся функцией `QTextCodec::codecForLocale()`.

Заметим также, что для перевода текста в UTF-8 есть очень удобное сокращение `QString::toUtf8()`.

9.4.4. Локализация

Локализация — это процесс адаптации к имеющимся локальным предпочтениям. Например, представления даты и времени в официальном для данного региона формате.

Подобного рода локализация осуществляется по средствам строкового метода `tr()`.

```
void Clock::setTime(const QTime &time)
 {
     if (tr("AMPM") == "AMPM") {
         // 12-х часовое время
     } else {
         // 24-х часовое время
     }
 }
```

В этом примере для США мы оставим как есть перевод "AMPM", и таким образом будем использовать 12-часовую ветку, но в Европе мы переведём его другим способом, что позволит использовать 24-хчасовую ветку.

Для локализации чисел используется класс **QLocale**.

Заметим также, что в MeeGo SDK возможно использовать разные изображения (и другие подключаемые ресурсы) для различных локалей. Однако, использовать локализованные изображения всё же не рекомендуется. Предпочтительно использовать общепонятные изображения, вместо того, чтобы опираться на странные метафоры и местные смыслы. Исключением являются изображения стрелок вперёд и назад (в большинстве случаев их необходимо перевернуть для Арабских и Израильских локалей).

9.4.5. Динамическая трансляция

Некоторым приложениям может потребоваться изменять язык в процессе работы программы. Чтобы оповестить виджеты об изменениях необходимо использовать метод `changeEvent()` для проверки того, что состоялось событие `LanguageChang` и обновлять отображаемый текст, используя фукнкцию `tr()`.

Например:

```
 void MyWidget::changeEvent(QEvent *event)
 {
     if (e->type() == QEvent::LanguageChange) {
         titleLabel->setText(tr("Document Title"));
         ...
         okPushButton->setText(tr("&OK"));
     } else
         QWidget::changeEvent(event);
 }
```

Все остальные события должны быть передано по средствам вызова реализации функции по умолчанию.

Список установленных переводчиков может измениться в ответ на событие `LocaleChange`, или приложение может предоставлять пользователю интерфейс, который даст пользователю возможность изменять текущий язык. Обработчик событий по умолчанию для подклассов **QWidget** реагирует на событие `QEvent:: LanguageChange`, и вызовет эту функцию при необходимости.

Событие `LanguageChange` выбросится, когда пользователь установит новый перевод (выполнится функция `QCoreApplication::installTranslator()`). Кроме того, другие компоненты приложения могут заставить виджеты обновить себя, отправив им событие `LanguageChange`.

9.5. Лабораторная работа № 7 «Работа с графикой и интернационализация»

9.5.1. Цель лабораторной работы

Целью лабораторной работы является знакомство с предоставляемыми MeeGo SDK средствами отображения двухмерной графики и интернационализации.

9.5.2. Введение

В рамках этой лабораторной работы мы создадим приложение, отображающее выбранную пользователем векторную графику (в виде svg-файла) и предоставляющее возможность динамически (в процессе работы программы) менять язык отображения интерфейса. В соответствии этими двумя функциями разделим наше приложение на две части: первая будет включать в себя работу с графикой; вторая — работу со средствами интернационализации. Для демонстрации двухмерной графики с использованием классов QPainter предоставим приложению возможность отображать выбранный файл поверх подложки, нарисованной с помощью этих интерфейсов.

План работы

Создание приложения включает в себя:

1. создание нового приложения;
2. создание проектного файла, продумывание общей структуры проекта;
3. создание ресурсного файла;
4. разработка интерфейса пользователя; описание слотов и событий интерфейса;
5. программирования основной части приложения;
6. интернационализация приложения.

Необходимые знания и навыки

Предполагается, что пользователь выполнил предыдущие лабораторные работы, имеет навыки программирования на языке C++ и прослушал соответствующую лекцию.

Необходимые программные и аппаратные средства

Предполагается, что для работы пользователя установлено MeeGo SDK версии не ниже 1.0, налажен процесс разработки приложений и установлен соответствующий набор библиотек, необходимых для кросскомпиляции.

9.5.3. Инструкция по выполнению работы

9.5.3.1. Создание нового приложения

- Войдите в *Qt Creator*
- Создайте новый проект (“File->New File or Project”)
- В открывшемся диалоге (Рис. 9.5.2) выберите пункт “Mobile Qt Application”
- Перейдите к следующему шагу, нажав кнопку “Choose”
- Введите название проекта (“lab_09”) и выберете его расположение
- Перейдите к следующему шагу, нажав кнопку “Next”
-

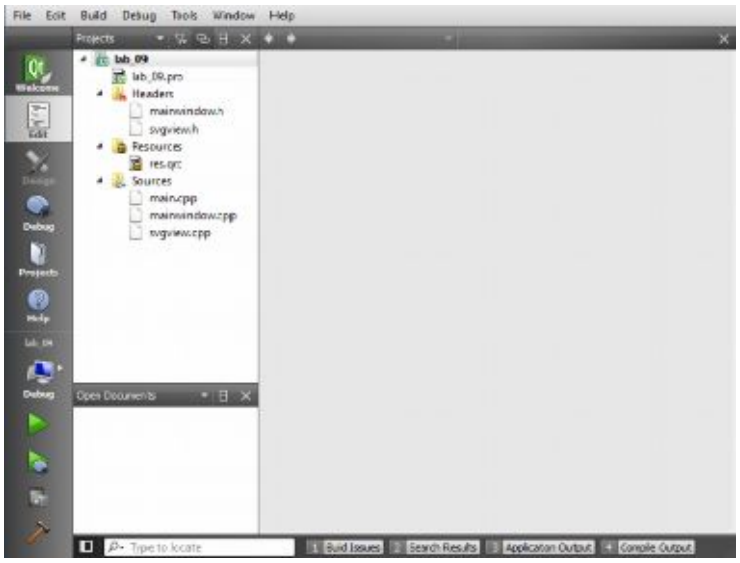


Рис. 9.5.1. Qt Creator.

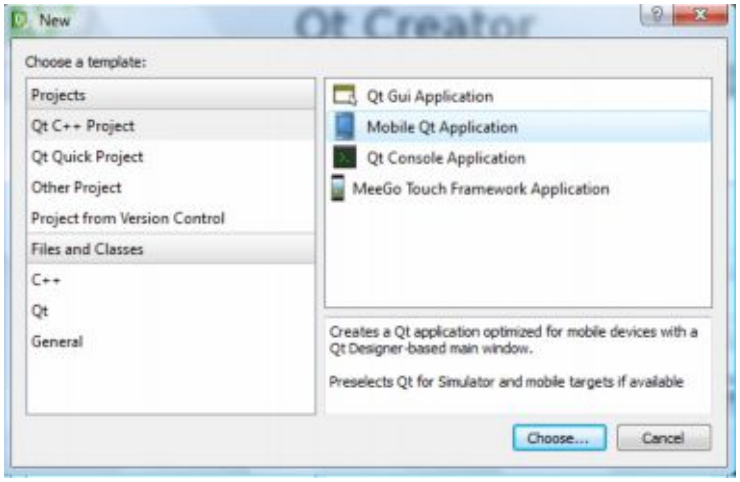


Рис. 9.5.2. Создание нового приложения.

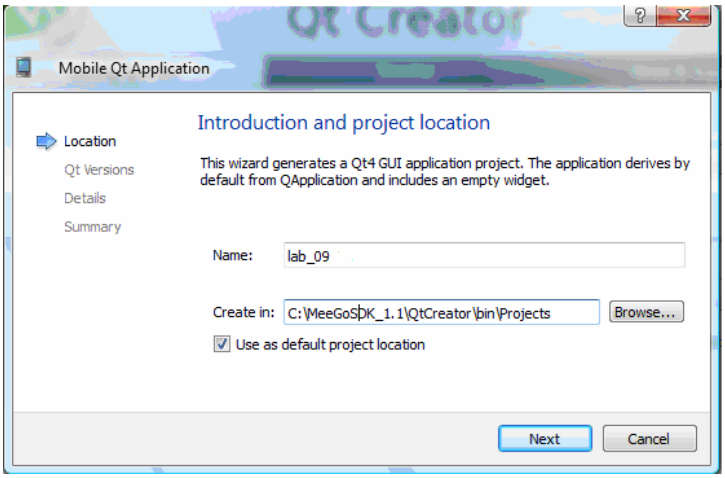


Рис. 9.5.3. Выбор имени и расположения проекта.

- Укажите целевые платформы для которых будет осуществляться кросскомпиляция (в данном случае это **meego-handset**, Рис. 9.5.4).

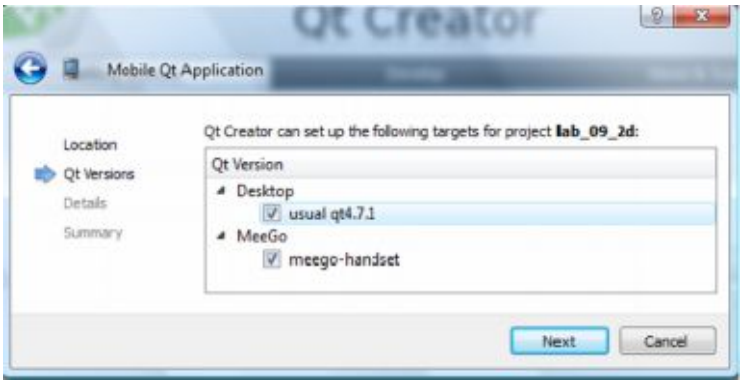


Рис. 9.5.4. Выбор целевых платформ.

- Оставляем нетронутыми названия генерируемых классов и снимаем галочку с Generate form (Рис. 9.5.5) — при разработке приложения **не будет** использован **Qt Designer**)

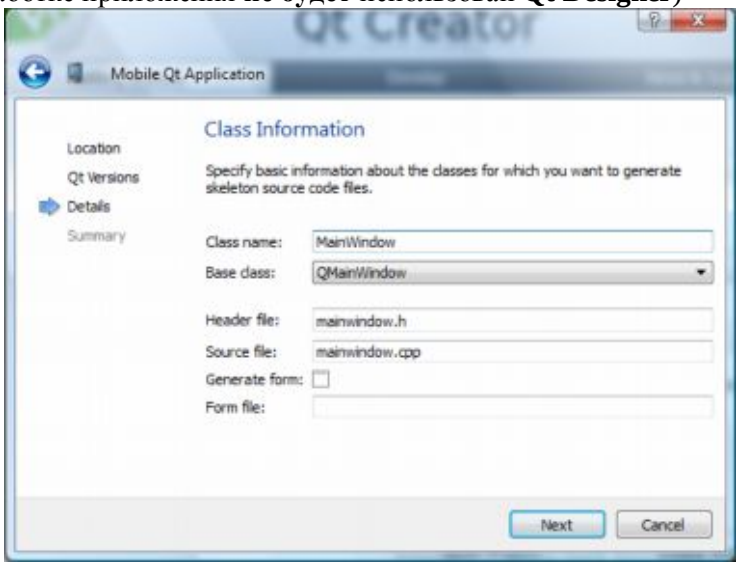


Рис. 9.5.5. Генерируемые классы.

- Добавьте проект под управление системой контроля версий, установленной на ПК — в данном случае, это GIT (Рис. 9.5.6)

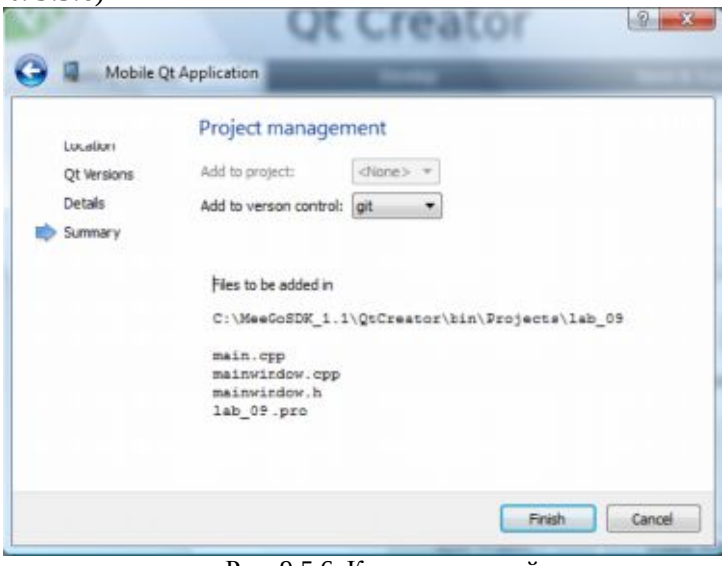


Рис. 9.5.6. Контроль версий.

9.5.3.2. Подготовка проектного файла

Для того чтобы наше приложение скомпилировалось необходимо правильно описать проектный файл (*.pro). Для этого отредактируем сгенерированный автоматически файл **lab_09.pro**.

```
QT += core gui svg
TARGET = lab_09
TEMPLATE = app
target.path=/usr/local/bin

INSTALLS = target
SOURCES += main.cpp\
                        mainwindow.cpp \
                        svgview.cpp
HEADERS  += mainwindow.h \
                         svgview.h
RESOURCES += res.qrc
TRANSLATIONS = lab09_en.ts \
                          lab09_ru.ts
```

Обратим внимание на строки **TRANSLATIONS** (забегая вперёд, данная строка определяет используемые в приложении языковые файлы) и **RESOURCES** (определяет ресурсный файл, в котором указываются все ресурсы, компилируемые в исполняемый файл**)**. Кроме того, обратим внимание на строку “QT=core gui svg” — в ней мы добавили необходимые библиотеки для работы с интерфейсом пользователя и векторной графикой в формате SVG.

В строках **SOURCES** и **HEADERS** фактически описана структура нашего приложения:

- в главном файле “main.cpp” осуществляется загрузка приложения и подключение нужного перевода;
- файл “mainwindow.cpp” описывает главное окно программы, а также слоты и события;
- “svgview.cpp” описывает класс виджета в котором будет отображаться svg-графика.

9.5.3.3. Подготовка ресурсного файла

В ресурсном файле мы указываем изображения, переводы, иконки и т. д. (все ресурсы), которые скомпилирует в конечный исполняемый файл. Для нашего проекта мы хотим подготовить файл содержащий:

- svg-изображение, появляющееся по умолчанию при запуске программы (файл “test.svg” — предварительно поместим его в созданную подпапку “files/images/” проекта);
- логотип MeeGo, которое мы загрузим используя QPainter (файл “files/images/meego_logo.svg”);
- и объектные файлы, содержащие перевод на русский и английский (“files/translations/lab09[_ru|_en].qm”).

Ресурсные файлы в Qt имеют расширение *.qrc, являются правильными xml файлами и могут быть созданы либо вручную, либо через *“File->New File->Qt->Qt Resource File”*. Если файл добавлялся и редактировался вручную, то необходимо не забыть добавить его под управление используемой системы контроля версий (для GIT — *git add res.qrc*).

Ресурсный файл нашего проекта (res.qrc) будет иметь вид:

```
<RCC>
    <qresource prefix="/">
        <file>files/images/test.svg</file>
        <file>files/images/meego_logo.jpeg</file>
        <file>files/translations/lab09_ru.qm</file>
        <file>files/translations/lab09_en.qm</file>
    </qresource>
</RCC>
```

9.5.3.4. Разработка интерфейса пользователя, описание слотов и событий

В проекте интерфейс будет создаваться вручную, без использования визуальных средств, предоставляемых Qt Designer. Это связано с тем, что несмотря на внешнюю простоту, работа с Qt Designer требует уверенного знания принципов и концепций лежащих в основе организации графического интерфейса средствами Qt. Рассмотрение данных концепций лежит вне предмета данной работы.

Интерфейс пользователя будет включать выпадающее меню, с использованием которого будет происходить работа с приложением. Для этих целей в файле “mainwindow.cpp” в конструкторе `MainWindow::MainWindow()` добавим строки:

```
this->fileMenu = new QMenu(tr("&File"), this);
        openAction = fileMenu->addAction(tr("&Open..."));
        openAction->setShortcut(QKeySequence("Ctrl+O"));
        quitAction = fileMenu->addAction(tr("E&xit"));
        quitAction->setShortcuts(QKeySequence::Quit);

this->viewMenu = new QMenu(tr("&View"), this);
        m_backgroundAction = viewMenu->addAction(tr("&Background"));
        m_backgroundAction->setEnabled(false);
        m_backgroundAction->setCheckable(true);
        m_backgroundAction->setChecked(false);
        m_outlineAction = viewMenu->addAction(tr("O&utline"));
        m_outlineAction->setEnabled(false);
        m_outlineAction->setCheckable(true);
        m_outlineAction->setChecked(true);

 this->windowBkgMenu = new QMenu(tr("&Background"), this);
        m_chessAction = windowBkgMenu->addAction(tr("&Chess"));
        m_chessAction->setCheckable(true);
        m_chessAction->setChecked(true);
        m_meegoAction = windowBkgMenu->addAction(tr("&MeeGo"));
        m_meegoAction->setCheckable(true);
        QActionGroup *bkgGroup = new QActionGroup(this);
            bkgGroup->addAction(m_chessAction);
            bkgGroup->addAction(m_meegoAction);

 this->langMenu = new QMenu(tr("&Language"), this);
        m_engAction = langMenu->addAction(tr("&English"));
        m_engAction->setCheckable(true);
        m_engAction->setChecked(true);
        m_rusAction = langMenu->addAction(tr("&Russian"));
        m_rusAction->setCheckable(true);
        QActionGroup *langGroup = new QActionGroup(this);
            langGroup->addAction(m_rusAction);
            langGroup->addAction(m_engAction);

        menuBar()->addMenu(this->fileMenu);
        menuBar()->addMenu(this->viewMenu);
        menuBar()->addMenu(this->windowBkgMenu);
        menuBar()->addMenu(this->langMenu);

        setCentralWidget(m_view);
        setWindowTitle(tr("SVG Viewer"));
```

Переменные **langMenu, windowBkgMenu, viewMenu** и другие, описаны в заголовочном файле “mainwindow.h”. Метод **QMenu::addAction** добавляет соответствующее действие в выпадающий пункт меню. Вызовы **openAction->setShortcut(QKeySequence("Ctrl+O"))** добавляют клавиатурные

сокращения для пунктов меню. Вызовы методов `setEnabled`, `setCheckable` и `setChecked` устанавливают доступность элементов меню и их исходные состояния. Вызовы методов `addMenu` прикрепляют описанные ранее пункты меню; `setWindowTitle` — устанавливает заголовок текущего окна.

Далее необходимо добавить описанным элементам меню обработчики с помощью системы слотов и сигналов Qt. Для этого в конструкторе допишем следующие строки:

```
connect(quitAction, SIGNAL(triggered()), qApp, SLOT(quit()));
connect(openAction, SIGNAL(triggered()), this, SLOT(openFile()));
connect(m_backgroundAction, SIGNAL(toggled(bool)), m_view,
                                        SLOT(setViewBackground(bool)));
connect(m_outlineAction, SIGNAL(toggled(bool)), m_view,
                                        SLOT(setViewOutline(bool)));
connect(bkgGroup, SIGNAL(triggered(QAction*)), this,
                                        SLOT(setWindowBackground(QAction*)));
```

Теперь необходимо описать соответствующие обработчики событий. Первым, в файле “mainwindow.cpp” опишем стандартный диалог открытия svg-файлов:

```
void MainWindow::openFile(const QString &path)
{
    QString fileName;

    if (path.isNull())
        fileName = QFileDialog::getOpenFileName(this, tr("Open SVG File"),
                m_currentPath, "SVG files (*.svg *.svgz *.svg.gz)");
    else
        fileName = path;

    if (!fileName.isEmpty()) {
        QFile file(fileName);

        if (!file.exists()) {
            QMessageBox::critical(this,
                                  tr("Open SVG File"),
                                  tr("Could not open file %1'.").arg(fileName));
            m_outlineAction->setEnabled(false);
            m_backgroundAction->setEnabled(false);
            return;
        }

        m_view->openFile(file);
        m_currentPath = fileName;
        setWindowTitle(tr("%1 - SVGViewer").arg(m_currentPath));

        m_outlineAction->setEnabled(true);
        m_backgroundAction->setEnabled(true);
        resize(m_view->sizeHint() + QSize(80, 80 + menuBar()->height()));
    }
}
```

Далее добавим выбор подложки.

```
void MainWindow::setWindowBackground(QAction *action) {
    if (action == m_chessAction){
        m_view->setBkgPaint(0);//chess
    } else if(action == m_meegoAction){
        m_view->setBkgPaint(1);//meego
    }
}
```

Эта функция в зависимости от действия будет вызывать с соответствующим параметром метод `setBkgPaint` класса **SvgView**.

9.5.3.5. Основная часть приложения

Итак, перейдём к основной части приложения. Главная функция (файл “main.cpp”) будет выглядеть следующим образом:

```
int main(int argc, char **argv)
{
    Q_INIT_RESOURCE(res);
    QApplication app(argc, argv);

    MainWindow window;
    if (argc == 2)
        window.openFile(argv[1]);
    else
        window.openFile(":/files/images/test.svg");
    window.show();

    return app.exec();
}
```

Вызов макроса **Q_INIT_RESOURCES** инициализирует ресурсы; ветвление “`if (argc == 2)`” позволяет запускать программу из консоли, с указанием в качестве параметра файла, который необходимо открыть. По умолчанию используется хранящийся в ресурсах svg-файл “:/files/images/test.svg” (**“:/”** означает ссылку на объект внутри ресурсного файла).

В файле svgview.cpp опишем конструктор виджета:

```
SvgView::SvgView(QWidget *parent)
    : QGraphicsView(parent)
    , m_svgItem(0)
    , m_backgroundItem(0)
    , m_outlineItem(0)
{
    setScene(new QGraphicsScene(this));
    setViewportUpdateMode(FullViewportUpdate);
    setBkgPaint(0);//set chess background
}
```

Наш виджет наследуется от класса **QGraphicsView** (смотреть заголовочный файл “svgview.h” в приложении к лабораторной работе), таким образом он наследует методы `setScene` — который прикрепляет новую графическую сцену к виджету, `setViewportUpdateMode`, устанавливающий способ обновления видимой области и прочие (полное описание методов можно найти в документации к Qt). Вызов метода `setBkgPaint` позволяет установить подложку по умолчанию. Он будет иметь следующий вид:

```
void SvgView::setBkgPaint(int idx) {
    QPixmap tilePixmap(64, 64);
    switch (idx) {
        case 0 : {//chess
                tilePixmap.fill(Qt::white);
                QPainter tilePainter(&tilePixmap);
                QColor color(220, 220, 220);
                tilePainter.fillRect(0, 0, 32, 32, color);
                tilePainter.fillRect(32, 32, 32, 32, color);
                tilePainter.end();
                break;
        }
        case 1 : {//meego
```

```
                tilePixmap.fill(Qt::white);
                QPainter tilePainter(&tilePixmap);
                QColor color(220, 220, 220);
                QImage *meego_img =
                                    new QImage(":/files/images/meego_logo.jpeg");
                tilePainter.drawImage(0,10,
                           meego_img->scaled(QSize(64,32),Qt::KeepAspectRatio));
                tilePainter.end();
                break;
        }
    }
    setBackgroundBrush(tilePixmap);
}
```

Здесь в каждом случае мы рисуем на тайлах (повторяющихся изображениях), размером 64x64 пикселей. Вызов метода `fill` с параметром `Qt::white` очищает подложку. Вызов конструктора `QPainter tilePainter(&tilePixmap)` позволяет создать новый объект, который осуществляет рисование на тайле. Метод `fillRect` заливают указанный прямоугольник; `drawImage` — рисует заданное изображение. Вызов метода **end** применяет изменения.

В следующей функции инициируется рисование подложки:

```
void SvgView::drawBackground(QPainter *p)
{
    p->save();
    p->resetTransform();
    p->drawTiledPixmap(viewport()->rect(), backgroundBrush().texture());
    p->restore();
}
```

Далее описываются слоты, которые позволяют пользователю установить окантовку и подложку для svg-файла.

```
void SvgView::setViewBackground(bool enable)
{
    if (!m_backgroundItem)
          return;

    m_backgroundItem->setVisible(enable);
}

void SvgView::setViewOutline(bool enable)
{
    if (!m_outlineItem)
        return;

    m_outlineItem->setVisible(enable);
}
```

И наконец, опишем метод `openFile`, в котором осуществляется вывод svg-графики из указанного файла:

```
void SvgView::openFile(const QFile &file)
{
    if (!file.exists())
        return;

    QGraphicsScene *s = scene();

    bool drawBackground = (m_backgroundItem ? m_backgroundItem->isVisible()
                                                                : false);
    bool drawOutline = (m_outlineItem ? m_outlineItem->isVisible() : true);
```

```
    s->clear();
    resetTransform();

    m_svgItem = new QGraphicsSvgItem(file.fileName());
    m_svgItem->setFlags(QGraphicsItem::ItemClipsToShape);
    m_svgItem->setCacheMode(QGraphicsItem::NoCache);
    m_svgItem->setZValue(0);

    m_backgroundItem = new QGraphicsRectItem(m_svgItem->boundingRect());
    m_backgroundItem->setBrush(Qt::white);
    m_backgroundItem->setPen(Qt::NoPen);
    m_backgroundItem->setVisible(drawBackground);
    m_backgroundItem->setZValue(-1);

    m_outlineItem = new QGraphicsRectItem(m_svgItem->boundingRect());
    QPen outline(Qt::black, 2, Qt::DashLine);
    outline.setCosmetic(true);
    m_outlineItem->setPen(outline);
    m_outlineItem->setBrush(Qt::NoBrush);
    m_outlineItem->setVisible(drawOutline);
    m_outlineItem->setZValue(1);

    s->addItem(m_backgroundItem);
    s->addItem(m_svgItem);
    s->addItem(m_outlineItem);

    s->setSceneRect(m_outlineItem->boundingRect().adjusted(-10, -10, 10, 10));
}
```

9.5.3.6. Интернационализация приложения

В учебных целях в следующем приложении реализуется перевод на русский и английские языки. Для того чтобы осуществить перевод приложения необходимо выполнить следующие шаги:

- Перейти в корень проекта.

```
> cd /home/user/qt/projects/lab_07
```

- Вызвать утилиту **lupdate** с указанием проектного файла.

```
> lupdate ./lab_07.pro
```

В результате вызова данной утилиты мы получим набор из двух файлов: *lab07_en.ts* и *lab07_ru.ts*, которые будут содержать маркированные вызовами функции `tr()` и макроса `QT_TR*_NOOP()` литерные тексты, содержавшиеся исходном C++ коде.

- Осуществить перевод маркированного литерного текста — для этого необходимо либо отредактировать получившийся .TS файл (который содержит правильный XML), либо использовать утилиту **Qt Linguist** (Рис. 9.5.6). Мы воспользуемся утилитой.
- В Qt Linguist необходимо открыть имеющиеся .TS файлы и осуществить перевод каждой фразы.
- Вызвать утилиту **lrelease** для того чтобы получить легковесный файл с сообщениями (.QM файл). Данный файл пригоден только для конечного использования.
- Скопируем получившиеся .QM файлы в пути, указанные в ресурсном файле (“./files/translations/lab07_en.qm” и “./files/translations/lab07_ru.qm”).

Подключим переведённые файлы, для этого в главном файле “main.cpp” добавим строки

```
QTranslator appTranslator;
appTranslator.load("lab09_en.qm",":/files/translations/");
app.installTranslator(&appTranslator);
window.setTranslator(&appTranslator);
```

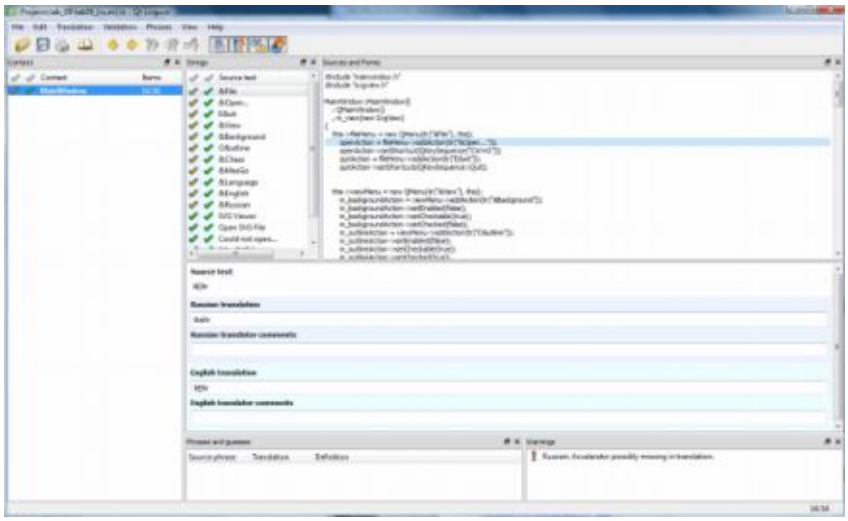


Рис. 9.5.6. Qt Linguist.

Метод `QTranslator::load()` загружает перевод; `QCoreApplication::installTranslator()` устанавливает выбранный переводчик; метод `MainWindow::setTranslator()` устанавливает перевод для главного окна.

- Добавим метод `MainWindow::retranslateUI()`, вызываемый когда пользователь инициирует переключение языка. В данном методе мы поочерёдно будем переназначать название каждого ранее маркированного элемента. В файл “mainwindow.cpp” добавим:

```
void MainWindow::retranslateUI() {
   this->fileMenu->setTitle(tr("&File"));
   this->viewMenu->setTitle(tr("&View"));
   this->windowBkgMenu->setTitle(tr("&Background"));
   this->langMenu->setTitle(tr("&Language"));
   setWindowTitle(tr("SVG Viewer"));
   this->m_backgroundAction->setText(tr("B&ackground"));
   this->m_outlineAction->setText(tr("O&utline"));
   this->m_chessAction->setText(tr("&Chess"));
   this->m_meegoAction->setText(tr("&MeeGo"));
   this->m_engAction->setText(tr("&English"));
   this->m_rusAction->setText(tr("&Russian"));
   this->openAction->setText(tr("&Open..."));
   this->quitAction->setText(tr("E&xit"));
}
```

- Соединим соответствующие слоты-события, связанные с переключением языка. Для этого добавим в файл “mainwindow.cpp” описание слота `setWindowLang`

```
void MainWindow::setWindowLang(QAction *action)
{
   if (action == m_engAction) {
      this->translator->load("lab09_en.qm",":/files/translations/");
   } else if(action == m_rusAction) {
      this->translator->load("lab09_ru.qm",":/files/translations/");
   }
   retranslateUI();
}
```

и соединим элементы в конструкторе

```
connect(langGroup, SIGNAL(triggered(QAction*)),
            this,          SLOT(setWindowLang(QAction*)));
```

9.6. Выводы

В этой лекции мы провели обзор возможностей MeeGo API для программирования 3D графики — были рассмотрели основные реализации OpenGL ES (1.X и 2.X), и их различия. Кроме того, изучили средства работы с векторной графикой (по средствам SVG формата) и простейшего рисования с помощью QPainter. Также были рассмотрены основные способы интернационализации, используемые в MeeGo API и их сценарии использования.

При выполнении лабораторной работе № 7 читатели могут получить практические навыки работы с векторной и двухмерной графикой, а также со средствами интернационализации, предоставляемыми MeeGo API.

9.7. Контрольные вопросы

1) Основным классом для работы с 2D графикой в MeeGo API является класс:
   1. QPainter
   2. QSvg
   3. QWidget
   4. QImage
2) Три основных класса, предназначенных для работы с двухмерной графикой в MeeGo API:
   1. QSvg -> QWidget -> QImage
   2. QWidget -> QPainter-> QPaintDevice
   3. QPainter -> QPaintEngine -> QPaintDevice
3) В качестве устройства вывода не может быть использован экземпляр класса:
   1. QWidget
   2. QImage
   3. QPixmap
   4. QPrinter
   5. QPaintEngine
4) Классом, предоставляющим абстрактное устройство вывода при работе с 2D графикой в MeeGo API, является класс:
   1. QSvgDevice
   2. QPaintDevice
   3. QPaintEngine
   4. QImageDevice
5) Следующий код:

```
void SimpleExampleWidget::paintEvent()
{
                QPainter paint(this);
                paint.setPen(Qt::blue);
                paint.drawText(rect(), Qt::AlignCenter, "Square");
};:
```

   1. Рисует квадрат, у которого снизу присутствует синяя подпись — “Square”.
   2. Рисует в центре прямоугольника, возвращаемого функцией *rect*, слово “Square”
   3. При вызове обработчика *paintEvent*, события рисования, изобразит по центру синий прямоугольник (свойства которого получаются с помощью функции *rect*), после чего ассоциирует с ним метку “Square”
6) Конструктор *QPaint (this)* предполагает рисование:
   1. На поверхности, заданной ранее при создании экземпляра класса QPaintEngine.
   2. На поверхности, заданной ранее при создании экземпляра класса QPaintDevice.
   3. На поверхности виджета, с которым ассоциирован в данный момент объект *this*.
7) При вызове метода *paint.drawText(rect(), Qt::AlignCenter, "Square")*, текст внутри прямоугольника будет выровнен:
   1. только по центру, по вертикали;

2. по левой стороне, по горизонтали;
3. по правой стороне, по горизонтали;
4. по центру, по вертикали и горизонтали одновременно;
5. только по центру, по горизонтали (выравниванием по вертикали занимаются флаги Qt::AlignTop, Qt::AlignBottom и Qt::AlignVCenter).

8) Рисование на пиксельных картах осуществляется с помощью методов:
   1. drawPixmap, drawImage, drawTiledPixmap;
   2. класса QPainterDevice
   3. класса QPainterPixelMap
9) Рисование примитивов осуществляется с помощью методов:
   1. QPainterDevice::draw*
   2. QPainter::draw*
   3. QPaint::drawPolygon
10) Рисование кривой Безье осуществляется с помощью метода:
   1. QPainter:: drawPolygon
   2. QPainter:: drawArc
   3. QPainter:: drawCubicBezier
11) Текущие установки шрифта регулируется вызовом функции:
   1. QPainter::setPen
   2. QPainter::setFont
   3. QPainter::setBrush
   4. QPainter::setFontOrigin
12) В чём разница между экземплярами классов *QBrush* и *QPen*:
   1. *QBrush* определяет цвет, или образец используемый для заливки; *QPen* определяет цвет, или пунктир, используемый для рисования линий и границ.
   2. *QPen* определяет цвет, или образец используемый для заливки; *QBrush* определяет цвет, или пунктир, используемый для рисования линий и границ.
   3. Оба класса предоставляют эквивалентные настройки рисования в контексте разных устройств вывода.
13) Метод, определяющий прозрачность используемой подложки:
   1. QPainter::setBrushMode
   2. QPainter::setPenMode
   3. QPainter::setBackgroundMode
   4. QPainter::setFontMode
14) Сохранение текущих установок шрифта во внутреннем стеке осуществляется вызовом функции:
   1. QPainter::saveFont
   2. QPainter::save
   3. QPainter::restore
   4. QPainter::restoreFont
15) Получение региона, по которому все остальные операции рисования осуществляют обрезание, осуществляется с помощью функции:
   1. QPainter::setRegion
   2. QPainter::killRegion
   3. QPainter::clipRegion
16) Какое аффинное преобразование не доступно методами стандартного интерфейса MeeGo API:
   1. Скручивание
   2. Отражение
   3. Поворот
   4. Сдвиг
   5. Растяжение
17) Область видимости задаётся вызовом метода:
   1. QPainter::setViewport
   2. QPainter::setWindow

3. QPainter::setMatrix

18) Работа с матрицей преобразований осуществляется с помощью стандартных методов:
    1. класса QMatrixPainter
    2. класса QMatrix
    3. класса QMatrixWindow
19) Для работы со сложной графикой и графикой, в которой требуется повторение различных преобразований, используется класс:
    1. QPainter
    2. QPainterPath
    3. QPaintGenerator
    4. QPaintSaver
20) Отрисовка сгенерированного пути (класс *QPainterPath*) инициируется вызовом метода:
    1. QPainter::pathFind
    2. QPainter::drawPath
    3. QPainter::pathSave
21) Официально в MeeGo поддерживается и предоставляются средства работы с векторной графикой в формате:
    1. VML
    2. SWF
    3. SVG
22) Три основных класса, через которые осуществляется работа со специальной векторной графикой:
    1. QSvgGenerator, QSvgRenderer, QSvgWidget
    2. QPainter, QPainterDevice, QPainterDriver
    3. QWidget, QImage, QGLWidget
23) Класс *QSvgRenderer* может быть использован для отрисовки SVG графики на любых экземплярах:
    1. классов QPainterDriver
    2. классов QPainterDevice, QPainterDriver
    3. классов QWidget, QImage, QGLWidget
    4. класса QPainterDevice
24) Класс *QSvgRenderer* может быть использован для отрисовки SVG графики на любых экземплярах:
    1. классов QPainterDriver
    2. классов QPainterDevice, QPainterDriver
    3. классов QWidget, QImage, QGLWidget
    4. класса QPainterDevice
25) Класс *QSvgGenerator* используется для
    1. отображения данных в формат SVG на экземплярах классов QWidget;
    2. создания изображений в формате SVG стандартными средствами интерфейсов *QPainter*;
    3. чтения данных из SVG файла.
26) Официально поддерживаемым средством работы с 3D графикой в MeeGo SDK является:
    1. VRML/X3D
    2. X.Org
    3. OpenGL ES
    4. Direct 3D, версия для мобильных устройств и нетбуков
    5. COLLADA
27) Для 3D графики в платформе MeeGo существует реализация:
    1. только спецификации OpenGL ES 1.x
    2. только спецификации OpenGL ES 2.x
    3. спецификаций OpenGL 4.1 и OpenGL ES 1.x
    4. спецификаций OpenGL ES 2.x и OpenGL ES 1.x
28) I18n, это:
    1. средства осуществляемые разработчиком на последнем этапе разработки программного продукта, направленные на адаптацию продукта для использования в других странах;
    2. сокращение от “Internationalization” — адаптация продукта для потенциального использования

практически в любом месте на начальных этапах разработки;
3. процесс всемирной экономической, политической и культурной интеграции и унификации взятый в контексте разработки программного обеспечения.

29) Маркировка целевых литерных текстов осуществляется вызовами:
1. функции *tr* и макроса *QT_TR_NOOP;*
2. функций transliterate_noop;
3. функции trans и макроса QTR.

30) *.TS файлы содержат:
1. уже переведённые тексты, в объектном виде;
2. тексты, автоматически переведённые на все языки;
3. тексты, маркированные в приложении для перевода.

31) *.QM файлы содержат:
1. уже переведённые тексты, в объектном виде;
2. тексты, автоматически переведённые на все языки;
3. тексты, маркированные в приложении для перевода.

32) Утилита *lrelease* предназначена для:
1. получения легковесного файла с сообщениями (.TS файл) из файла, содержащего маркированные в приложении литерные тексты (.QM файл), пригодный только для конечного использования;
2. получения легковесного файла с сообщениями (.TS файл) из файла, содержащего маркированные в приложении литерные тексты (.QM файл), пригодный как для конечного использования, так и для использования в утилите QLinguist;
3. получения легковесного файла с сообщениями (.QM файл) из файла, содержащего маркированные в приложении литерные тексты (.TS файл), пригодный только для конечного использования;
4. автоматического распознавания и перевода литерных текстов, располагающихся в исходных кодах программы.

33) Для перевода текстов используется утилита MeeGo API:
1. Qt Creator
2. Qt Linguist
3. Qt MeeGo
4. Qt Translator
5. lupdate

34) Вопрос №34 *[слайд 20]*: Следующий пример:

```
QCString str = "OK";
QPushButton *ok = new QPushButton( tr(str), this );
```

1. не скомпилируется;
2. скомпилируется, но не запустится на целевом устройстве;
3. скомпилируется, запустится, но не пометит литерный текст “OK” как текст, для перевода;
4. зависит от реализации транслятора. Есть вероятность, что литерный текст “OK” пометится для перевода.

35) Русскоязычное приложение следует разрабатывать:
1. без использования лишних функций, вставляя литерные тексты на русском языке прямо в текст программы;
2. таким образом, чтобы все литерные тексты были на английском и маркировались вызовами *tr*, а далее осуществлялся перевод с помощью специальных утилит SDK;
3. с использованием кодировок ISO, а именно ISO-8859-5, чтобы исключить проблемы с интернационализацией.

36) Для трансляции текстов используется утилита MeeGo API:
1. Qt Linguist
2. Qt MeeGo
3. lupdate
4. Qt Translator

5. Qt Designer

37) Используемые переводы указываются:
   1. в отдельном ресурсном файле *.res, который подключается при запуске программы;
   2. в коде приложения в главном заголовочном файле main.h;
   3. в проектном файле в разделе TRANSLATIONS.
38) Основным классом, через который осуществляется интернационализация в приложении, является класс:
   1. QTranslator
   2. QLinguist
   3. QMeeGo
   4. QL10n
39) Динамическая интернационализация предназначена для
   1. изменение отображаемой информации при изменении системной локали;
   2. редактирования ресурсных файлов приложения в процессе работы программе;
   3. изменения отображаемой приложением информации при изменении используемого языка, средствами приложения в процессе работы программы.
40) Динамическая интернационализация осуществляется
   1. при обработке события QEvent::LanguageChange и установке всех значений заново;
   2. автоматическим образом;
   3. в организации отдельного потока, в котором в бесконечном цикле проверяется, не изменился ли используемый язык.
41) Локализация может использоваться для:
   1. отображения текстов и надписей на языке пользователя;
   2. представления даты и времени в официальном для данного региона формате;
   3. изменения даты и времени в соответствии с часовым поясом пользователя;
42) Использование локализации в MeeGo API основано на:
   1. использовании специального набора интерфейсов QLocalize;
   2. использовании стандартных средств интернационализации;
   3. использовании платных сторонних утилит.

Список литературы

1. MeeGo Core API (http://apidocs.meego.com)
2. MeeGo Core API — I18n (http://apidocs.meego.com/1.1/core/html/qt4/internationalization.html)
3. MeeGo Core API — OpenGL ES (http://apidocs.meego.com/1.1/core/html/categories/Graphics.html)
4. MeeGo Core API — QPainter (http://apidocs.meego.com/1.1/core/html/qt4/painting.html)
5. MeeGo Core API — SVG (http://apidocs.meego.com/1.1/core/html/qt4/qtsvg.html)
6. Qt Documentation (http://doc.qt.nokia.com/4.7/index.html)
7. MeeGo wiki (http://wiki.meego.com)
8. Статья “Forgot a tr()?” (http://doc.trolltech.com/qq/qq03-swedish-chef.html)

10. MeeGo API: работа с данными пользователя. QtMobility

MeeGo API: Data Mgmt. Content Framework [Tracker], Context Framework [ContextKit], Package Manager [PackageKit]. Примеры: файловые менеджеры, устройства для просмотра изображений и т. п.

10.1. Введение

Эта лекция представляет собой краткий обзор возможностей, предоставляемых MeeGo SDK по управлению данными пользователя. В её рамках мы, затронем возможности MeeGo по хранению данных приложения (**Publish and Subscribe**), управлению контактами (библиотека **Contacts**), а также работе с визитными карточками (формата **vCard**) и прочей информации в формате **Versit**.

Кроме того, мы рассмотрим способы получения системной информации.

10.2. Qt Mobility

- Проект Qt Mobility — это часть Qt, предоставляющая набор Qt библиотек, которые хорошо известны из мира мобильных устройств, в частности, телефонов. При этом MeeGo API использует функции этого фреймворка как на мобильных устройствах, так и на нетбуках.

В MeeGo API версии 1.1 реализована версия 1.0 фреймворка Qt Mobility, которая, к сожалению, не включает работу с камерой. Релиз содержит следующий набор интерфейсов:

- **Contacts —** интерфейс позволяющий разработчику управлять контактными данными пользователя.
- **Location —** интерфейс определения местоположения, предоставляющий библиотеки для распространения и получения данных о местоположении пользователя.
- **Messaging —** интерфейс отправки сообщений, который может быть использован для отправки/получения сообщений, а также поиска и их сортировки. Кроме того, интерфейс предоставляет возможность выбрать предпочтительный способ обмена сообщениями и формат его отображения, создать новое сообщение, или ответить на существующее. API поддерживает SMS, MMS, работу с электронной почтой в форматах MIME и TNEF.
- **Multimedia —** интерфейс для работы с мультимедиа данными предоставляет возможности воспроизведения, записи и управления медиа-контентом.
- **Publish and Subscribe —** интерфейс хранения данных позволяет приложениям хранить, читать и получать уведомления об изменениях хранящихся данных.
- **Sensors —** интерфейс работы с датчиками предоставляет доступ к датчикам и охватывает как датчики высокого уровня, например, ориентацию экрана (портрет и пейзаж), так и низкого (работающих в реальном времени), например, датчик скорости передачи данных.
- **Service Frameworks —** библиотеки, предназначенные для расширения функциональность MeeGo API с использованием сервисов.
- **System Information —** интерфейс, предоставляющий набор библиотек для сбора информации об устройствах и их возможностях.
- **Versit —** управление документами в формате Versit, например, vCard.

Обзор библиотек **Location** уже был сделан выше. В сегодняшней лекции мы остановим своё внимание на интефейсах **Publishing and Subscribe**, **System Information**, **Contacts** и **Versit**.

10.2.1. Подключение

Для работы с Qt Mobility необходимо предпринять три шага.

```
// Главный файл проекта
#include <QSystemInfo>
QTM_USE_NAMESPACE
```

```
int main(int argc, char *argv[])
{
   QSystemInfo si;// библиотека Qt Mobility
}
```

1. Добавить заголовки используемых модулей Qt Mobility (в примере выше — QSystemInfo);
2. Использовать макрос **QTM_USE_NAMESPACE**;
3. Добавить в проектный файл строчки:

```
CONFIG += mobility
MOBILITY += systeminfo
```

10.2.2. Хранение данных (Publish and Subscribe)

Интерфейс **Publish and Subsrcibe** предназначен специально для того, чтобы дать приложениям удобный способ хранения настроек приложения. Он реализован в виде специального хранилища, который позволяет хранить и считывать элементы данных, а также подписываться на их изменения.

Пространство значений (англ. Value Space) объединяет различные источники иерархические даных в единую непротиворечивую модель. Концептуально, пространство значений является иерархическим деревом, в котором каждый узел или лист может, при необходимости, содержать значение типа **QVariant**. Сериализованная версия простого примера пространства значений может выглядеть следующим образом:

```
/Device/Buttons = 3
/Device/Buttons/1/Name = Menu
/Device/Buttons/1/Usable = true
/Device/Buttons/2/Name = Select
/Device/Buttons/2/Usable = false
/Device/Buttons/3/Name = Back
/Device/Buttons/3/Usable = true
```

Доступ к существующим значениям в пространстве значений можно получить через класс **QValueSpaceSubscriber**. Этот класс предоставляет возможности чтения значений и получения уведомлений об изменениях заданного пути и навигации по пространству. Новые значения добавляются по средствам класса **QValueSpacePublisher**.

Уведомления об изменениях моделируется интересным образом. Например, при изменении **/Device/Buttons/1/Name —** изменённым пометится также **/Device/Buttons/1** и т. д. вверх по дереву. Это позволяет, например, подписчикам на **/Device/Buttons/1** получать уведомления, когда изменяется любое свойство кнопки.

Пример использования класса **QValueSpacePublisher**:

```
// Добавление значений

QValueSpacePublisher  publisher([const QUuid & uuid,] “/My”);

publisher.setValue(“Buttons”, “1”);
publisher.setValue(“Buttons/1/Name”, tr(“Button 1”));
publisher.setValue(“Buttons/1/Enabled”, false);
publisher.sync() ;

// Удаление добавленных значений

publisher.resetValue (“Buttons”);
publisher.sync();

// QValueSpaceSubscriber("/My/Buttons").value() == QVariant();
// QValueSpaceSubscriber("/My/Buttons/1").value() == QVariant();
```

```
// QValueSpaceSubscriber("/My/Buttons/1/Name").value() == QVariant();
```

Второй пример использования интерфейса:

```
MyWidget::MyWidget()
{
   ...
   // Получение значения
   QValueSpaceSubscriber *buttons1 = new QValueSpaceSubscriber(“/My”);
   QValueSpaceSubscriber buttons2(“/My/Buttons”);

   buttons2.value() == buttons1->value(“Buttons”); // верно

   buttons1->cd(“Buttons”); // перейти в поддиректорию

   // Подписка на изменения значений
   QObject::connect( buttons1, SIGNAL(contentsChanged()), this,
                                                SLOT(buttonInfoChanged()));

   // Изменения
   publisher.setValue(“/My/Buttons/1/Name”, tr(“Вызовет сигнал”));
   publisher.setValue(“/My/Buttons/1”, tr(“Вызовет сигнал”));
   publisher.setValue(“/My”, tr(“Не вызовет сигнал”));
}
```

На текущий момент для платформы MeeGo доступно хранение данных только в GConf. В следующих версиях планируется реализация хранилища данных в общей памяти (англ. Shared Memory Layer). В хранилище GConf возможно хранение только ограниченного множества типов данных — это Bool, Int, Double, String, StringList и List. Все остальные типы конвертируются в BASE64 строки и, по необходимости, деконвертируются обратно прозрачным для пользователя образом.

10.2.3. Получение системной информации (System Information)

Этот интерфейс предоставляет набор классов для получения системной информации.

Интерфейс подмодуля Qt Mobility **System information** позволяет возвращать информацию по следующим категориям:

- **О версиях и спектре поддержке ПО устройства.** Например, об операционной системе и микропрограмме, а также версиях WebKit, Qt и Service Framework.
- **Возможностях аппаратной платформы.** В том числе список поддерживаемого оборудования на устройстве. Характеристики включают такие элементы как: камера, Bluetooth, GPS, FM-радио и т.д.
- **Сеть.** Состояние сетевого соединения, а также тип сети, например GSM, CDMA, Ethernet и др.
- **Информация о дисплее.**
- **Информация об устройстве:**
  - состояние батареи;
  - профиль (тихий, вибрация, нормальный и т. д.);
  - сим-карта;
  - устройства ввода (кнопки, клавиатура, QWERTY, сенсорный экран, мультитач и т. д.).

Основные классы Qt Mobility System Information:

- **QSystemDeviceInfo** — доступ к информации об устройстве;
- **QSystemDisplayInfo** — доступ к информации об экране;
- **QSystemInfo** — доступ к различной общей информации;
- **QSystemNetworkInfo** — доступ к сетевой информации;

- **QSystemScreenSaver** — доступ к скринсейверу и затемнению;
- **QSystemStorageInfo** — доступ к информации о памяти и диске.

Примеры

```
MyWidget::MyWidget()
{
   ...
   // Заряд батареи:
   QSystemDeviceInfo *di;
   qDebug() << "Заряд батареи = " << di->batteryLevel();

   // Подписка на уведомления об изменении уровня заряда:
   QObject::connect( di, SIGNAL(batteryLevelChanged(int)),
                  this, SLOT(updateBatteryStatus(int)) );

   // Информация о производителе и продукте:
   qDebug() << "Производитель: " << di->manufacturer();
   qDebug() << "Название продукта: " << di->productName();
   ...
}
```

Некоторая информация хранится в побитовом виде. В приведенном выше коде рассмотрено определение имеющихся устройств ввода, использующее битовые флаги.

```
QSystemDeviceInfo::InputMethodFlags methods = di-> inputMethodType();
QStringList inputs;
if((methods & QSystemDeviceInfo::Keys)){
            inputs << "Keys";
}
if((methods & QSystemDeviceInfo::Keypad)) {
            inputs << "Keypad";
}
```

Доступные возможности могут быть найдены путем тестирования на их наличие.

```
QSystemInfo::Feature feature;
QSystemInfo si;
qDebug() << "Поддержка Bluetooth" <<
si.hasFeatureSupported(QSystemInfo::BluetoothFeature);
```

10.3. Управление персональной информацией (PIM)

Управление персональной информацией (англ. "Personal Information Management", сокр. PIM) — это набор интерфейсов Qt Mobility, связанных с управлением контактной и бизнес информацией в форматах **Versit**. Оно включает в себя стандартный набор интерфейсов по работе адресной книгой Qt Mobility Contacts и возможность по сериализации адресной книги в форматы данных **vCard** и **iCalendar**.

10.3.1. Управление контактами (Qt Mobility Contacts)

Контактная информация хранится в специальном хранилище данных, функциональные возможности которого доступны через менеджера. В рамках интерфейса контакт моделируется как коллекция различных деталей. Каждая деталь соответствует конкретному определению (или шаблону), который может быть расширяемым или, иначе, изменяемый с помощью клиентов. Индивидуальные контакты могут быть связаны друг с другом и их отношения хранятся отдельно от самих контактов. Используя синхронные и асинхронные интерфейсы можно получать, изменять, или удалять информацию о контактах, их детальные определения и их связях.

Контактная информация хранится в классах-контейнерах. Эти классы не являются производными от **QObject**, могут быть использованы в списках, не имеют родителей, не выбрасывают сигналы, и так далее. Они представляют собой данные, которыми можно манипулировать и извлекать из менеджера.

Контакты (Contact)

Под «Контактом» имеется в виду цифровое представление лица, группы, или организации, которое хранится платформо-специфичным образом. Информация, касающаяся одного контакта может храниться в разных хранилищах данных и каждая единица данных (или деталь), может как относиться, так и не относиться к конкретному контексту в котором она верна.

Контакт может включать в себя семантически идентичные детали, верные в разных контекстах. Например, контакт может иметь телефонный номер, который имеет отношение к контексту его «дома» и другой телефонный номер, который имеет отношение к его «работе».

Деталь (Detail)

Под «Деталью» имеется в виду единая, сплочённая единица информации, которая хранится в контакте. Как объяснялось ранее, она справедлива для определенного контекста, или набора контекстов, и соответствует конкретному определению. Деталь может иметь ассоциированные с ней специальные метаданные, такие как её подтип, контекст и границы, определяемые пользователем метаданные, также как и ограничения доступа, которые могут применяться к детали (например, «только для чтения», «неудаляемое» и т. д.).

Отношения (Relationships)

Контакты могут участвовать в отношениях с другими контактами. Подробная информация о любых таких отношениях хранится менеджером, который содержит контакт. Есть несколько стандартных типов отношений, поддерживаемых стандартной схемой и произвольные типы отношений, которые допускается, если менеджер поддерживает эту функцию. Одним из важных отношений является «групповое отношение» — членство контакта в группе может быть смоделировано как если бы контакт группы участвовал в отношении hasMember с контактом. За работу с отношениями отвечает класс QContactRelationship.

Доступ к данным

Доступ к контактам осуществляется по средствам интерфейсов модуля Manager. Менеджер предоставляет доступ к хранилищам данных, поддерживаемым платформой. Каждое хранилище может поддерживать разные возможности (например, возможность хранения определенных типов данных, способность к родному фильтру на различные детали, или детали различных определений, обеспечение механизмов блокировки, предоставление изменений информации и т.д.), которые сообщаются менеджером по запросу. Кроме того, менеджер предоставляет доступ к деталям определений, контактам и отношениям, хранящимся в различных типах хранилищ данных, в платформо- и хранилище-независимом образах.

Для получения списка всех имеющихся менеджеров может быть использована функция `QContactManager:: availableManagers()`. Каждый менеджер идентифицируется URI. URI состоит из имени менеджера, любых соответствующих параметров, которые были использованы при создании его экземпляра и его версии. Хотя имя менеджера определяет плагин, который обеспечивает функциональность, вы не можете гарантировать, что данные, доступные через один менеджер будет доступна через другой с таким же именем (например, если один параметр говорит плагину хранить и извлекать контактную информацию из сим-карты, а другой из локального хранилища).

Для работы с менеджером рекомендуется использовать ассинхронный интерфейс доступа к данным, т. е. такой интерфейс, использующий неблокирующие, асинхронные запросы. Для большинства приложений рекомендуется использование именно этот способ.

Асинхронный интерфейс включает в себя следующие классы, полученные из класса **QContactAbstractRequest – QContactLocalIdFetchRequest** ,
QContactFetchRequest,

QContactSaveRequest,
QContactRemoveRequest,
QContactDetailDefinitionFetchRequest, **QContactDetailDefinitionSaveRequest**,
QContactDetailDefinitionRemoveRequest, **QContactRelationshipFetchRequest**,
QContactRelationshipSaveRequest
и
QContactRelationshipRemoveRequest.

Более того, разработчик может реализовывать собственные движки, используя классы **QContactManagerEngine** и **QContactManagerEngineFactory**.

Основные классы

Основные классы, используемые в модуле:

- **QContact** — представление контакта из адресной книги;
- **QContactAbstractRequest** — механизм асинхронных запросов менеджеру, если таковые поддерживаются;
- **QContactDetail** — представление единственной, полной детали о контакте;
- **QContactFilter** — предоставляет возможность выделить контакты доступные через **QContactManager**;
- **QContactManager** — менеджер доступа к контактной информации;
- **QContactRelationship** — отношение один-к-одному между двумя объектами.

Различные наследники класса **QContactDetail**, которые обязана иметь каждая реализация Qt Mobility:

- **QContactAddress** — адрес контакта;
- **QContactAnniversary** — юбилей контакта;
- **QContactAvatar** — аватар контакта;
- **QContactBirthday** — день рождение контакта;
- **QContactDisplayLabel** — отображаемое имя контакта (возможно синтезированное)
- **QContactEmailAddress** — email контакта;
- **QContactFamily** — имена членов семьи контакта;
- **QContactGender** — пол контакта;
- **QContactGeoLocation** — геоположение контакта;
- **QContactGlobalPresence** — агрегация текущей информации о контакте, возможно синтезированной, или предоставленной движком;
- **QContactGuid** — глобальный уникальный идентификатор контакта, используемый для синхронизации с другими хранилищами данных;
- **QContactName** — имя контакта;
- **QContactNickname** — никнейм контакта;
- **QContactNote** — заметка о контакте;
- **QContactOnlineAccount** — онлайн аккаунт, который контакт использует для связи с друзьями и семьёй;
- **QContactOrganization** — организация к которой принадлежит контакт;
- **QContactPhoneNumber** — телефонный номер;
- **QContactPresence** — информация о присутствии в сети для онлайн аккаунта контакта;
- **QContactRingtone** — рингтон, ассоциированный с контактом;
- **QContactTag** — тэг, ассоциированный с контактом;
- **QContactThumbnail** — эскиз, используемый при выводе контактов списком;
- **QContactTimestamp** — содержит время и дату создания и последнего изменения данных;
- **QContactType** — описывает тип контакта;
- **QContactUrl** — URL контакта.

Каждый из этих подклассов обеспечивает доступ к информации, хранящейся в областях, которые могут иметь определенные ограничения, как перечислено в соответствующей схеме.

Интерфейс предоставляет возможность получить контакт, указав его уникальный идентификатор, или используя **QContactFilter**, который соответствует контакту, или контактам, которые необходимо выбрать. Следующие классы являются производными от **QContactFilter** и позволяют точно и гибко выбирать контакты по различным критериям:

- **QContactChangeLogFilter** — фильтр на основе времени последнего изменения детали;
- **QContactDetailFilter** — фильтр на основе значений деталей;
- **QContactIntersectionFilter** — фильтр, пересекающий результаты работы других фильтров;
- **QContactInvalidFilter** — фильтр выдающий пустое множество результатов;
- **QContactLocalIdFilter** — фильтр на основе списка идентификаторов контакта;
- **QContactRelationshipFilter** — фильтр основанный на критериях отношений;
- **QContactUnionFilter** — фильтр, объединяющий результаты работы других фильтров.

Кроме того, возможно запросить отсортированные значения, передав **QContactSortOrder**, или список команд сортировки менеджеру.

Примеры

Получение списка доступных менеджеров.

```
// Получение списка доступных менеджеров
QStringList availableManagers = QContactManager::availableManagers();
QMap<QString, QString> params;
QContactManager manager;
QString mngrURI;

if ( -1 != availableManagers.indexOf(“nokiasim”) ) {
// менеджер контактов сим-карты
      params.insert("store", "ADN"); // способ хранения данных
      mngrURI = QContactManager::buildUri(“nokiasim”, params);
} else {
      mngrURI = QContactManager::buildUri(availableManagers.last());
}

manager = QContactManager::fromUri(mngrUri);
```

Добавление полей контакту.

```
QContact c = manager->contact(myId); // Получаем контакт

QContactPhoneNumber mobile = c.detail<QContactPhoneNumber>(); // Добавляем поля
QContactPhoneNumber home = c.detail<QContactPhoneNumber>();

mobile.setNumber(“8-952-555-55-55”); // Записываем телефоны
home.setNumber(“555-55-55");

QStringList mobile_contexts;  mobile_contexts << “home” << “mobile”;
QStringList home_contexts;    home_contexts << “home”;

mobile.setContexts(mobile_contexts); // Добавляем контексты
home.setContexts(home_contexts);

c.saveDetail(&mobile); // Сохраняем изменения
c.saveDetail(&home);

// Получаем список номеров с контекстом «дом»
QList<QContactPhoneNumber> homePhones =
   contact.details<QContactPhoneNumber>("Context", “home");
```

Определение отношений между контактами.

```
// Определить новое отношение между контактами
```

```
QContact c1 = manager->contact(id1); // Получаем контакт1
QContact c2 = manager->contact(id2); // Получаем контакт2

QContactRelationship r = new QContactRelationship();
   r.setFirst(id1);
   r.setSecond(id2);
   r.setRelationshipType(“doctor”);

manager->saveRelationship(r);

// Использование собственных отношений
QList<QContactId> therapists =
   c2.relatedContacts(“doctor", QContactRelationship::Second);
```

```
// Контакт groupContact имеет тип «группа»
groupContact.type()== QContactType::TypeGroup // верно

// Получить членов группы
QList<QContactId> groupMembers =
   groupContact.relatedContacts(QContactRelationship::HasMember,

QContactRelationship::Second);

// Получить группы, в которых состоит контакт
QList<QContactId> contactGroups =
   contact.relatedContacts(QContactRelationship::HasMember,
                                          QContactRelationship::First);
```

10.3.2. Сериализация в формат данных Versit

В MeeGo SDK возможно работать не только с данными в формате Qt Mobility Contacts – существует возможность работы с повсеместно используемыми электронными визитными карточками и другими «бизнес-данными» в формате Versit (Versit® зарегистрирована как торговая марка консорциума IMC (Internet Mail Consortium)).

Основным форматом Versit является vCard – формат файлов для обмена электронными визитными карточками. Файл в формате vCard состоит из набора записей, каждая из которых содержит информацию одной визитной карточки. Запись может содержать имя, адрес, номера телефонов, URL, логотип, видео/аудио фрагменты и другую информацию.

Формат vCard (или Versitcard), разработан в 1995 консорциумом Versit, в который вошли Apple Computer, AT&T (позднее Lucent), IBM и Siemens. В декабре 1996 все права на формат перешли к Internet Mail Consortium.

В MeeGo API имеется возможности по работе с версиями 2.1 и 3.0 данного формата. Версия 2.1 получила поддержку в большинстве почтовых клиентов. Версия 3.0 описана в RFC 2425 и RFC 2426. Обычно, файлы vCard имеют расширение .vcf.

Пример визитной карточки в формате vCard (версии 3.0), содержащей данные об одной персоне:

```
BEGIN:VCARD
VERSION:3.0
N:Василий;Пупкин
FN:Пупкин Василий
ORG:Компания по производству денег
PHOTO;VALUE=URL;TYPE=GIF:http://www.example.com/dir_photos/my_photo.gif
TEL;TYPE=WORK,VOICE:8-904-555-55-55
TEL;TYPE=HOME,VOICE: 8-904-666-66-66
```

```
ADR;TYPE=HOME:;;пр. Невский 55;55;Санкт-Петербург;55555;Россия;Земля
LABEL;TYPE=HOME:Невский пр-т 55-55\nСанкт-Петербург, индекс: 555555\nРоссия
EMAIL;TYPE=PREF,INTERNET:vasiliy@example.com
REV:20080424T195243Z
END:VCARD
```

Кроме того, консорциумом разработан целый ряд форматов для передачи других «бизнес-данных» — например, формат передачи календарных данных iCalendar, TO-DO записей (vTODO), журнальных отметок (vJOURNAL). Как видно из следующего примера, все они в основе имеют один и тот же принцип:

```
BEGIN:VCALENDAR
VERSION:2.0
PRODID:-//hacksw/handcal//NONSGML v1.0//EN
BEGIN:VEVENT
UID:uid1@example.com
DTSTAMP:19970714T170000Z
ORGANIZER;CN=John Doe:MAILTO:john.doe@example.com
DTSTART:19970714T170000Z
DTEND:19970715T035959Z
SUMMARY:Bastille Day Party
END:VEVENT
END:VCALENDAR
```

Основные классы

Для работы с документами формата Versit применяются следующие классы:

- **QVersitDocument** — документ в формате Versit;
- **QVersitProperty** — элемент данный Versit-документа;
- **QVersitReader** — чтение Versit-документов;
- **QVersitWriter** — запись Versit-документов;
- **QVersitContactImporter** — импорт контактов из формата Versit в формат QContact;
- **QVersitContactExporter** — экспорт контактов из формата QContact в формат Versit.

Примеры

Следующие примеры демонстрируют процесс чтения карточек в формате vCard и их импорта в формат QContact, из которого совершается обратный экспорт и последующая запись.

```
QByteArray input =     “BEGIN:VCARD\r\nVERSION:3.0\r\n”
           “N:Василий;Пупкин\r\nEND:VCARD\r\nEND:VCARD”;

//Импорт данных в формат QContact
QVersitReader *reader = new QVersitReader(input);
QList<QVersitDocument> inputDocuments = reader.results();

QVersitContactImporter importer;
if (!importer.importDocuments(inputDocuments))
   return;

QList<QContact> contacts = importer.contacts();
manager::saveContacts(&contacts);//сохраняем контакты
```

Для разбора визитной карточки в формате vCard, или календаря в формате iCalendar, полученных из устройств ввода/вывода, может быть использован класс **QVersitReader** (на выходе мы получим список документов в формате **QVersitDocument**).

```
// Экспорт данных обратно в Versit

QVersitContactExporter exporter;
```

```
if (!exporter.exportContacts(contacts, QVersitDocument::VCard30Type))
   return;
QList<QVersitDocument> outputDocuments = exporter.documents();
```

В завершении процесса, мы можем использовать класс **QVersitWriter** для записи данных обратно в экземпляр класса **QByteBuffer**.

```
QVersitWriter *writer = new QVersitWriter(input);
                        writer-> startWriting(outputDocuments);
```

10.4. Лабораторная работа № 8 «Приложение для рассылки SMS»

10.4.1. Цель лабораторной работы

На примере решения задачи автоматической рассылки приглашений на день рождения по контактам адресной книги с фильтром по диапазону возрастов контактов научиться пользоваться инструментами Qt Mobility.

10.4.2. Введение

В этой лабораторной работе будет рассмотрен пример мобильного приложения для автоматической рассылки приглашений на день рождения по контактам адресной книги с фильтром по диапазону возрастов контактов.

10.4.3. 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.

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

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

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

Функция отправляет сообщение *message* каждому контакту в списке.

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

Функция отображает список контактов в окне вывода.

4. `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;
for each (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. На нетбуке с MeeGo Netbook нет штатных средств для отправки SMS и нет такого аккаунта, поэтому, запуская приложение на таком нетбуке, отправить SMS не удастся. Но на более подходящей платформе рассмотренное приложение должно работать полноценно.

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.

10.4.4. 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.

10.5. Выводы

В этой лекции мы провели обзор возможностей, предоставляемых MeeGo API для работы с данными пользователей — способы их хранения, записи и сериализации в формат vCard. Кроме котого, были рассмотрены способы получения системных данных и механизм хранения настроек. В лабораторной работе рассмотрен пример рассылки SMS-сообщений по списку контактов.

10.6. 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

Список литературы

1. MeeGo Core API (http://apidocs.meego.com)
2. MeeGo Core API — System Information (http://apidocs.meego.com/1.1/core/html/qtmobility/systeminfo.html)
3. MeeGo Core API — Publish and Subscribe (http://apidocs.meego.com/1.1/core/html/qtmobility/publ-subs.html)
4. MeeGo Core API — Versit (http://apidocs.meego.com/1.1/core/html/qtmobility/versit.html)
5. MeeGo Core API — Contacts (http://apidocs.meego.com/1.1/core/html/qtmobility/contacts.html)
6. MeeGo Core API — Messaging (http://apidocs.meego.com/1.1/core/html/qtmobility/messaging.html)
7. MeeGo Core API — Personal Information Management (http://apidocs.meego.com/1.1/core/html/categories/Personal_Information_Management.html)
8. MeeGo wiki (http://wiki.meego.com)
9. Hendy's Qt Mobility Development (http://qt-mobility.blogspot.com/)
10. Qt 4.7.1 и Qt Mobility 1.1.0 / Qt Software / Хабрахабр (http://habrahabr.ru/blogs/qt_software/107959/)

11. MeeGo API: мультимедиа, датчики, сообщения, коммуникация между приложениями

MeeGo API: Media Services. Media Framework [GStreamer], Camera [Gstreamer plug-in], Codecs [Gstreamer plug-in], Audio [PulseAudio], UPnP [UPnP]. Пример подключения к IVI камеры видеонаблюдения, реализация сервисов по выявлению движущихся объектов и передачи фотоизображений или видеопотока.

11.1. Введение

В этой лекции мы проведём обзор возможностей, предоставляемых MeeGo SDK для работы с мультимедиа, сообщениями и аппаратными датчиками. Кроме того, кратко будут рассмотрены основные способы коммуникации между приложениями.

11.2. Мультимедиа

В MeeGo реализован собственный фреймворк для раобты с мультимедиа данными — Qt Multimedia Framework (Qt MMF). Этот фреймворк предоставляет разработчику удобные и гибкие интерфейсы для воспроизведения медиа файлов, организации плейлистов и слайд-шоу, а также записи аудио и работы с радио. К сожалению, на момент написания этого текста (релиз SDK версии 1.1) работа с камерой была недоступна несмотря на наличие интерфейсов **QCamera**.

На уровне операционной системы, данные возможности реализуются по средствам кроссплатформенного набора библиотек **GStreamer**, но пользователь волен реализовывать недостающую функциональность самостоятельно.

GStreamer — мультимедийный фреймворк, написанный на языке программирования C и использующий систему типов GObject. GStreamer является «ядром» мультимедийных приложений, таких как видеоредакторы, потоковые серверы и медиаплееры. В изначальный дизайн заложена кроссплатформенность; GStreamer работает на Unix-подобных системах, а также на Microsoft Windows, OS/400 и Symbian OS. GStreamer предоставляет биндинги для других языков программирования таких, как Python, C++, Perl, GNU Guile и Ruby. GStreamer является свободным программным обеспечением, с лицензией GNU LGPL [11].

Фреймворк MMF пришёл на смену интерфейсам Phonon, которые, в свою очередь, тоже использует в качестве драйвера библиотеку GStreamer.

В отличие от других интерфейсов Qt Mobility, MMF не является частью пространства имён QtMobility, а значит, при его использовании нет необходимости объявлять пространство имён.

11.2.1. Аудио

Начнём с примера. Рассмотрим приложение, использующее класс Audio Recorder для записи звука.

При записи источника звука есть ряд вещей, которые можно контролировать. Например, мы можем выбрать формат кодирования файлов — MP3 или Ogg Vorbis, или, например, выбрать другой источник входа. Пользователь может изменить битрейт, количество каналов, качество и частоту дискретизации.

Итак, первым делом устанавим источник и рекордер объекта. Объект **QAudioCaptureSource** создается и используется для инициализации объекта **QMediaRecorder**.

```
audiosource = new QAudioCaptureSource;
capture = new QMediaRecorder(audiosource);

capture->setOutputLocation(QUrl("test.raw"));
```

Далее найдём поддерживаемые кодеки:

```
QStringList codecs = capture->supportedAudioCodecs();
capture->setAudioCodec( codecs.last() );
```

Теперь достаточно начать запись по средствам вызова метода:

```
capture->record();
```

и остановить её вызовом:

```
capture->stop();
```

Фреймворк использует обобщённый класс **QMediaPlayer** в качестве плеера для всех медиа файлов. При этом воспроизводить файлы очень просто: необходимо создать экземпляр плеера, передать имя файла, установить громкость, используемые параметры и инициировать игру. Всё!

```
QMediaPlayer *player = new QMediaPlayer;
...
player->setMedia(QUrl::fromLocalFile("test.raw"));
player->setVolume(50);
player->play();
```

Файл не обязан быть локальным, это может быть указатель на удаленный ресурс. Также с помощью **QMediaPlaylist** класса из этого интерфейса можно играть список локальных или удаленных файлов. **QMediaPlaylist** класс поддерживает построения, управления и воспроизведения плейлистов.

```
player = new QMediaPlayer;

playlist = new QMediaPlaylist(player);
playlist-> append(QUrl("http://example.com/myfile1.mp3"));
playlist-> append(QUrl("http://example.com/myfile2.mp3"));
...
playlist->setCurrentPosition(1);
player->play();
```

Для управления плейлистом существуют обычные функции управления (которые на самом деле реализованы в виде слотов): previous (предыдущий файл), `next` (следующий), `setCurrentPosition` (установка текущей позиции) и `shuffle` (случайный режим).

11.2.2. Видео

Кроме воспроизведения аудио, интерфейс предоставляет возможности по отображению видео потоков. Есть определённые преимущества этого интерфейса (применительно к Qt). Во-первых, разработчик может реализовывать основные функции мультимедиа минимальными силами, в основном, потому что они уже реализованы. Второе — источник звука не обязан быть локальным файлом устройства — это может быть удалённый потоковый файл. А значит нет необходимости в написании кода, который, определив тип источника (локальный/удалённый), будет скачивать его при помощи интерфейсов **Qt Network** и уже дальше передавать на воспроизведение — всё делается парой методов.

Рассмотрим аналогичный пример по воспроизведению набора видео-файлов.

```
player = new QMediaPlayer;

playlist = new QMediaPlaylist(player);
playlist->append(QUrl("http://example.com/myclip1.mp4"));
playlist->append(QUrl("http://example.com/myclip2.mp4"));
...
widget = new QVideoWidget(player);
widget->show();

playlist->setCurrentPosition(1);
player->play();
```

Переход с аудио к видео требует пары изменений в исходный код. Для воспроизведения, код видео-плейлиста необходимо изменить таким образом, чтобы включить еще один новый мобильный класс **QVideoWidget**. Этот класс позволит нам управлять видео-ресурсом, а также с помощью

механизма сигналов и слотов мы сможем контролировать яркость, контраст, оттенок, насыщенность и режим просмотра видео (полноэкранный/ оконный).

```
videoWidget = new VideoWidget(this);
player->setVideoOutput(videoWidget);
...
playlistModel = new PlaylistModel(this);
playlistModel->setPlaylist(playlist);
```

11.2.3. Радио – пример

Лучше всего работу с радио иллюстрирует следующий пример. Далее, **QRadioTuner** является базовым классом, который для каждой конкретной платформы является основой для управления радио.

```
//Воспроизведение радио
QRadioTuner radio;
if (radio->isBandSupported(QRadioTuner::FM)) {
//      Радио поддерживает воспроизведение FM
        radio->setBand(QRadioTuner::FM);
        radio->setFrequency(104);
        radio->setVolume(100);
        radio->start();
}
```

Фреймворк предоставляет простейшие функции по работе с FM-радио, в том числе: настройка, громкость, запуск, остановка и различное другое управление процессом воспроизведения.

11.2.4. Слайдшоу – пример

В качестве ещё одной демонстрации гибкости и удобства фреймворка рассмотрим его возможности по отображению слайдшоу. При их отображение используется уже знакомыq нам класс для создания плейлистов **QMediaPlaylist**. Кроме того, используется класс **QMediaImageViewer** для вывода изображений.

Следующий код выполняется в контексте любого производного QObject. Ключевое слово **this**, переданное в конструктор **QMediaImageViewer** определяет, что родителем данного объекта будет упомянутый контекст.

```
viewer = new QMediaImageViewer(this);

display = new QVideoWidget;
viewer->setVideoOutput(display);
display->show();

playlist = new QMediaPlaylist(this);
playlist->setMediaObject(viewer);
playlist->setPlaybackMode(QMediaPlaylist::Loop);
playlist->addMedia(image1);
playlist->addMedia(image2);
playlist->addMedia(image3);

viewer->setTimeout(5000);
viewer->play();
```

11.3. Использование аппаратных датчиков

Часто разработчику мобильных приложений на платформе MeeGo приходится снимать показания с датчиков. 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.

Каждый датчик в системе имеет свой *идентификатор* и *тип*. По умолчанию SDK предоставляет набор интерфейсов для работы с наиболее распространёнными типами датчиков, а именно:

- акселерометр (**QAccelerometer**);
- магнитометр (**QMagnetometer**);
- компас (**QCompass**);
- датчик освещённости (**QAmbientLight**);
- датчик ориентации (**QOrientation**);
- датчик поворота (**QRotation**);
- датчик близости (**QProximity**);

и общие классы:

- **QSensor** — единственный аппаратный сенсор;
- **QSensorFilter** — эффективная возможность для обратных вызовов и асинхронных уведомлений об обновлениях датчиков;
- **QSensorReading** — чтение показаний.

В связи с тем, что различные аппаратные платформы предоставляют различный набор датчиков и, к тому же, некоторые датчики могут быть недоступны — вы можете использовать следующие функции для получения всех доступных типов датчиков на платформе:

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

Далее, получить идентификаторы всех датчиков заданного типа возможно используя следующую конструкцию:

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

Типичный жизненный цикл датчика включает в себя:

1. Создание экземпляра **QSensor**, или одного из его подклассов.

```
QSensor sensor( sensor_ids.last() );
```

2. Установки в соответствии с требованиями приложения.
3. Запуск процесса получения значений.

```
sensor.start();
```

4. Чтение датчика данных, используемых приложением.

```
QSensorReading *reading = sensor.reading();
qreal x = reading->property("x").value<qreal>();
```

5. Отказ от получения значений.

```
sensor.stop();
```

Пример: получение показаний акселерометра:

```
QAccelerometer sensor; sensor.start();
QAccelerometerReading sensor_reading = sensor.reading();
qreal x = sensor_reading->x();
sensor.stop();
```

Подпись на уведомления об изменениях показаний датчика используя стандартные сигналы/слоты Qt:

```
MyWidget::MyWidget() {
...
 QObject::connect(
    sensor, SIGNAL( readingChanged() ),
    this,      SLOT( myFunction() );
...
}
```

В некоторых случаях обновления могут приходить слишком часто и будут вызывать снижение производительности системы. В этом случае более правильным будет использовать производные класса **QSensorFilter**, например, **QAccelerometerFilter**. Для его использования необходимо наследоваться и реализовывать виртуальный метод **filter**, в котором необходимо добавить логику работы в случае обновления датчика.

```
bool QAccelerometerFilter::filter(QAccelerometerReading * reading)[pure virtual]
```

11.4. Работа с сообщениями

В MeeGo SDK существует специальный обобщённый интерфейс для работы практически со всеми типами сообщений — отправки SMS, сообщений электронной почты, мгновенных средств обмена данными, собственных форматов данных. Библиотека **Qt Messaging** обеспечивает поиск и сортировку сообщений, их отправку и получение. Кроме того, она предоставляет специальное хранилище из которого, используя средства всё той же библиотеки, возможно выборочно отображать существующие сообщения, создавать новые и отвечать на существующие.

Основным классом при работе с сообщениями является класс **QMessage**, представляющий собой сообщение любого типа (SMS, MMS, MIME Email, TNEF Email и др.). Управление сообщениями организует класс **QMessageManager** — основной интерфейс для хранения и поиска сообщений, папок и ячеек в хранилище сообщений. Каждое сообщение хранится в логической ячейке хранилища сообщений — за это отвечает класс **QMessageAccount**. При этом сообщения могут группироваться по папкам (класс QMessageFolder) и фильтроваться с использованием класса **QMessageFilter**. Наконец, за специфическую для типа сообщения отправку отвечает класс **QMessageService**.

Таким образом, библиотека предоставляет унифицированный доступ к данным сообщений на всех устройствах, включая одновременный доступ к сообщениями данных несколькими приложениями, и независимость от механизма, используемого для хранения сообщений данных на устройстве.

Приведём несколько простых примеров.

Создание SMS:

```
QMessageAddress to (MessageAddress::Phone,“89055555555”);

QMessage sms;
                  sms.setBody("Hello MeeGo! " );
                  sms.setType(QMessage::SMS);
                  sms.setTo(to);
```

Регистрация и сохранение сообщения в системе:

```
QMessageManager().addMessage( *sms );
```

Отправка сообщения:

```
QMessageService().send( sms );
```

Фильтрация сообщений:

```
QMessageFilter includeFilter(QMessageFilter::byTimeStamp(minimumDate,
QMessageDataComparator::GreaterThanEqual));
QMessageFilter excludeFilter(QMessageFilter::byTimeStamp(maximumDate,
QMessageDataComparator::GreaterThanEqual));
QMessageFilter outgoingFilter(QMessageFilter::byStatus(QMessage::Incoming,
QMessageDataComparator::Excludes));

// Поиск сообщений, содержащих адреса для исключения
service.queryMessages(outgoingFilter & excludeFilter);
```

11.5. Коммуникация между процессами (IPC)

Часто при разработке приходится осуществлять коммуникацию нескольких приложений, например, вызов методов одного приложения другим и т. д. Подобное взаимодействие возможно осуществить тремя основными способами:

- через локальные сокеты
- через общую память
- через шину данных D-BUS.

В следующем примере *Приложение А* передаёт *Приложению В* изображение тремя различными способами.

11.5.1. Локальные сокеты

Передающее приложение (А) будет играть роль клиента, а принимающее — сервера. Детали реализации опущены.

Приложение А:

```
//Создаёт локальный сокет
QLocalSocket socket;
//Соединяется с сервером с указанием его имени
socket.connectToServer("img_server" );
//Передаёт данные
//Закрывает сокет
socket.close();
```

Приложение В:

```
//Создаёт слушающий сокет
QLocalServer server;
//Настраивает сокет для принятия подключений:
server.listen("img_server" );
//Получает изображение через сокет
//Закрывает сокет
```

Основные недостатки такого подхода:

1. необходимо понимание концепции сокетов;
2. приложение вынужденно принимает вид клиент-серверного.

11.5.2. Общая память

Рассмотрим тот же пример с использованием общей памяти.

Приложение А:

```
//Получаем доступ к общей памяти
QSharedMemory memory ("key1 " );
//Подготавливаем данные
QBuffer buffer;
buffer.open(QBuffer::ReadWrite);
QDataStream out(&buffer);
out << image;
int size = buffer.size();

if (!sharedMemory.create(size)) {
   qDebug() << "Unable to create shared memory segment.";
   return;
}

//Защищаем память от записи
memory.lock();

//Записываем данные
char *to = (char*)memory.data();
const char *from = buffer.data().data();
memcpy(to, from, qMin(memory.size(), size));

//Снимаем защиту
memory.unlock();
```

Приложение B:

```
//Получаем доступ к общей памяти QSharedMemory memory ( “key” );
//Ждём пока в памяти не появятся данные,
//проверяем готовность методом memory.isAttached ();

//Подготавливаем данные
QBuffer buffer;
QDataStream in(&buffer);
QImage image;

//Читаем из общей памяти
//Защищаем память от записи
memory.lock();
buffer.setData((char*)memory.constData(), memory.size());
buffer.open(QBuffer::ReadOnly);
in >> image;//например,

//Снимаем защиту
memory.unlock();
//Освобождаем память
memory.detach();
```

Главные минусы такого подхода:

1. приложение не может получить уведомление о записи в общую память, используя сигналы и слоты;
2. приложение вынуждено захватывать память (метод `memory.lock()`) на время записи;
3. приложения должны заранее знать ключ, по которому записывается и считывается значение из общей памяти.

11.5.3. Qt D-Bus

Шина данных D-Bus – это самый цивилизованный способ коммуникации приложений. На данный момент, в ОС Linux именно на его плечах лежит организация связи между приложениями (в том числе он используется в связке с аппаратной шиной HAL).

Необходимость появления программных шин первоначально возникла в средах GNOME и KDE, где множеству настольных приложений был необходим удобный механизм связывания. Первоначально подобная коммуникация осуществлялась по средствам CORBA, SOAP или XML-RPC. Например, в среде GNOME D-BUS заменил Bonobo, который, в свою очередь, был основан на CORBA, но в связи с зависимостью от GObject, дальше среды GNOME не использовался.

D-BUS – это механизм IPC (InterProcess Communication), предоставляющий собой шину для передачи сообщений, ну а если быть точнее, то сразу несколько шин. Первая и самая главная — системная шина, она создается уже при старте демона D-BUS, и с ее помощью происходит общение различных демонов. Она хорошо защищена от посторонних, и пользовательские приложения, хоть и могут подключаться к ней, все же будут значительно ограничены в том, какие сообщения они смогут туда посылать (в то же время они могут многое «услышать»).

Реальная же рабочая лошадка D-BUS — сессионная шина, создаваемая для любого пользователя, авторизующегося в системе. Для каждой такой шины запускается отдельная копия демона, и именно посредством нее будут общаться приложения, с которыми работает этот пользователь.

Для идентификации и объектов используются пути, именуемые в стиле Unix. Так, например, сам D-BUS доступен по адресу “/org/freedesktop/DBus”.

Каждый объект может поддерживать один или более интерфейсов, которые представлены здесь в виде именованных групп методов и сигналов — аналогично интерфейсам Glib, Qt или Java.

Именуются они привычным для ОО-программистов образом, например рассматриваемый D-BUS экспортирует интерфейс “org.freedesktop.DBus”.

Каждое сообщение D-BUS, передаваемое по шине, имеет своего отправителя и своего получателя, их адреса называются путями объектов, поскольку D-BUS предполагает, что каждое приложение состоит из набора объектов, а сообщения пересылаются не между приложениями, а между объектами этих самых приложений.

Однако, объектов и интерфейсов недостаточно для реализации некоторых интересных возможностей, и D-BUS также предусматривает концепцию сервисов. Сервис — уникальное местоположение приложения на шине и пространство имён. При запуске приложение регистрирует один или несколько сервисов, которыми оно будет владеть до тех пор, пока самостоятельно не освободит, до этого момента никакое другое приложение, претендующее на тот же сервис, занять его не сможет. Именуются сервисы аналогично интерфейсам, а сам D-BUS экспортирует, соответственно, сервис “org.freedesktop.DBus”.

Сервисы делают доступной еще одну функцию — запуск необходимых приложений в случае поступления сообщений для них. Для этого должна быть включена автоактивация, и в конфигурации D-BUS за этим сервисом должно быть закреплено одно приложение. Тогда D-BUS сможет его запустить при появлении сообщения.

После закрытия приложения ассоциированные сервисы также разрегистрируются, а D-BUS посылает сигнал о том, что сервис закрыт. Другие приложения могут получать такие сигналы и соответствующим образом реагировать.

Сообщения в D-BUS бывают четырех видов:

- вызовы методов,
- результаты вызовов методов,
- сигналы,
- ошибки.

Первые предназначены для выполнения методов над объектами, подключенными к D-BUS; посылая такое сообщение, вы выдаете задание объекту, а он после обработки обязан возвратить вам либо результат вызова, либо ошибку через сообщения соответствующих типов. Сигнальные же сообщения, как им и полагается, ничуть не заботятся о том, что и как делается объектами, они вольны воспринимать их как угодно (равно как и не получать их вовсе).

Чтобы сообщение достигло определённого объекта, необходим способ сослаться на объект. Во многих языках программирования это реализуется с помощью указателя. Однако, эти указатели реализуются как адреса памяти, относящиеся к локальному адресному пространству приложения, и не могут быть переданы от одного приложения другому.

Поэтому в D-Bus у каждого объекта своё, уникальное имя, которое выглядит как путь в файловой системе. Например, объект может быть именован как “org/kde/kspread/sheets/3/cells/4/5”. Предпочтительны имена, которые несут какую-либо смысловую нагрузку, тем не менее, разработчики могут выбрать и имя /com/mycompany/c5yo817y0c1y1c5b, если это имеет смысл для их приложения.

Имена объектов находятся в пространствах имён, чтобы обеспечить разграничение разных программных модулей. Пространствам имён обычно даётся префикс, специфичный для разработчика, например “/org/kde”.

Кроме того, после подключения к шине приложение может настроить какие сообщения оно желает получать, путем добавления «масок». Маски представляют собой наборы правил для сообщений, которые будут доставляться приложению, фильтрация может основываться на интерфейсах, путях объектов и методах. Таким образом, приложения будут получать только то, что им необходимо, проблемы доставки в этом случае берет на себя D-BUS.

Реализация рассмотренного выше примера (опуская детали) будет следующей:

Приложение А:

```
//Создаётся интерфейс DBus
QDBusInterface remoteApp("com.example.ImageSaver","/ImageSaver/Operations",
                                                   "org.example.RPN.ImageSaver");
//Передаём SMS в сервис
```

```
remoteApp.call("PushData", img_data );
```

Приложение B:

```
//Адаптер описывается в формате XML и траслируется с помощью утилиты
//qdbusxml2cpp. При этом
//указываем название интерфейса  – "org.example.RPN.ImageSaver"

//Регистрируем сервис на шине
registerService ("com.example.ImageSaver" );

//Создаём его экземпляр
ImageSaver img_saver;

//Регистрируем интерфейс на шине
registerObject ("com.example.ImageSaver","/ImageSaver/Operations",* img_saver);
```

11.6. Лабораторная работа № 9 «Пример получения и записи данных от видеокамеры»

11.6.1. Цель лабораторной работы

На примере задачи о получении и записи данных (кадров) от видеокамеры системы видеонаблюдения научиться использовать компонентны MeeGo API по работе с видео.

11.6.2. Инструкция для выполнения лабораторной работы

Рассмотрим задачу получения данных (кадров) от видеокамеры системы видеонаблюдения и сохранения его в формате JPG.

Будем считать, что в нашем распоряжении имеется мобильное вычислительное устройство работающее под управлением ОС MeeGo. Но также мы будем учитывать возможность замены ОС другим Linux. Драйвера камеры интегрированы в ядро MeeGo и она доступна как устройство /dev/video0, это верно практически для любой современной USB вебкамеры. Решая рассматриваемую задачу, можно пойти разными путями:

1. Использовать непосредственно интерфейс video4linux2 (v4l2), предоставляемый современными ядрами Linux. Плюсы – наиболее полный доступ к аппаратным возможностям и отсутствие зависимостей, минусы – невозможность использовать вне Linux, необходимость работы на достаточно низком уровне, изучение документации.
2. Использовать готовую консольную утилиту, способную выполнить нашу задачу, Она может быть очень маленькой, как luvcview, а может очень мощной, как mplayer. Это самый быстрый путь получения результата, но самый трудный в плане развития.
3. Использовать библиотеку gStreamer (см. лекцию 11), входящую в состав MeeGo, она более удобна для программиста и богата дополнительными возможностями.
4. Использовать QT Phonon (см. лекцию 11), который, в свою очередь, обращается к gStreamer. Вдобавок к преимуществам последнего получаем интерфейс QT. Существенный минус – Phonon нет в MeeGo, его придется переносить самостоятельно.
5. Использовать Qt Multimedia Framework (см. лекцию 11), он присутствует в MeeGo и находится в состоянии интенсивного развития.

К счастью, после недолгого поиска удалось обнаружить утилиту, сделанную по сценарию 1 и полностью решающую нашу задачу – http://www.twam.info/linux/v4l2grab-grabbing-jpegs-from-v4l2-devices. Поэтому достаточно просто скачать ее исходный текст. Поскольку утилита небольшая по размеру, мы соберем ее прямо на целевой платформе, как мы бы сделали это в обычном Linux. Надо установить зависимости – компилятор и jpeg-библиотеку –

```
yum install gcc libjpeg-devel
```

затем скомлировать код

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

и проверить, что утилита работает

```
./v4l2grab -d /dev/video0 -o image.jpg
```

в image.jpg должен появиться кадр с камеры. В случае дешевой камеры кадр может быть плохого качества, так как камере перед работой требуется «разогрев» несколько секунд, а захват происходит сразу.

Утилита v4l2grab оказалась слишком простым решением, поэтому давайте все-таки попробуем решить ту же задачу другим способом – используя библиотеку gStreamer. Ее основная концепция состоит в использовании конвейеров (pipelines, широко используемое в Linux понятие), элементами которых могут быть источники, приемники и фильтры, последние совмещают функциональность первых двух. Например, видеокамера может быть источником, кодек или конвертер фильтром, а окно на экране – приемником, на практике цепочки могут ветвиться и быть гораздо длиннее. На конвейере обрабатывается мультимедиа-поток. Главная задача программиста в этой модели – составление правильного конвейера для своей задачи, объединение его со своим приложением с помощью сообщений и обратных вызовов и при необходимости – создание собственных элементов, если не хватает штатных.

Обратимся к нашему приложению gstgrab. В нем мы решаем 3 подзадачи –

- построение конвейера, извлекающего видеопоток из вебкамеры,
- получение кадра из элемента конвейера в наш обработчик и
- последующее сохранение кадра в файл в формате JPG.

Сначала мы создаем конвейер функцией `gst_pipeline_new()` и его элементы функцией `gst_element_factory_make()`. У всех объектов должны быть уникальные имена, при создании элементов мы также указываем их тип. Потребуется три элемента: источник-видеокамера имеет в gStreamer тип «v4l2src», в качестве приемника выступит фиктивный приемник «fakesink». Приемник фиктивный, потому что мы не хотим обрабатывать весь видеопоток, нам достаточно получить доступ к одному кадру. Если бы мы хотели не только захватить кадр, но и параллельно отобразить поток на экране, то нужно было бы разветвить поток на два и отображать на экран через настоящий приемник «ximagesink».

Если мы попробуем запустить конвейер всего из этих двух элементов («v4l2src» и «ximagesink»), то получим сообщение о несовпадении форматов. Это напоминает о том, что gStreamer – библиотека достаточно низкого уровня и многие вещи программист должен делать сам. Третьим элементом будет фильтр «ffmpegcolorspace», конвертирующий видео в формат RGB888 640x480.

Добавляем все элементы в конвейер функцией `gst_bin_add_many()` и связываем их в цепочку вызовами `gst_element_link_many()` и `gst_element_link_filtered()`. Теперь конвейер настроен и готов к запуску.

Чтобы получить кадр от фиктивного приемника, нужно назначить обработчик на его сигнал «handoff». Это делается фунцией библиотеки GTK `g_signal_connect()`, сигналы GTK используются в gStreamer и этот механизм является прообразом слотов и сигналов в QT. Наш обработчик называется `capture_callback()` и одним из агрументов он получает ссылку на буфер с данными «fakesink». В обработчике нужно сохранить буфер в файл и сообщить gStreamer о том, что мы больше не нуждаемся в его услугах.

Для сохранения буфера в файл используется функция `create_jpeg()`, в ней – типичный пример применения библиотеки libjpeg, которая даже в файл записывает сама, программисту не надо выделять для этого дополнительный буфер. Но зато очень важно, чтобы параметры сжатия строго совпадали с тем форматом видео, который был задан фильтром «ffmpegcolorspace». У библиотеки нет никакой дополнительной информации, чтобы проверить корректность входного буфера. Результатом ошибки будет испорченное изображение или даже сбой всей программы. Это еще одно напоминание, что мы ведем работу на достаточно низком уровне.

Все готово, и можно запускать конвейер. Обратите внимание, что это происходит в два этапа – сначала мы задаем конвейеру нужное состояние через `gst_element_set_state()`, а затем запускаем цикл обработки событий GTK `g_main_loop_run()` и он уже управляет работой конвейера.

Приложение gStreamer также соберем на целевой платформе. Устанавливаем зависимости

```
yum install gcc libjpeg-devel gstreamer-jpeg gstreamer-ffmpeg
```

При сборке мы будем использовать утилиту pkg-config, которая предоставляет компилятору набор параметров, необходимых для подключения больших, сложных библиотек. GTK и gStreamer безусловно являются сложными, поэтому вызов gcc выглядит так:

```
gcc gstgrab.c  -o gstgrab -ljpeg `pkg-config --cflags --libs gtk+-2.0 gstreamer-
interfaces-0.10`
```

Подключаем камеру и проверяем результат в image.jpg, запустив

```
./gstgrab
```

11.6.3. Задания для самостоятельной работы

1. Интегрировать v4l2grab в QT-приложение
2. Создать QT-обертку для v4l2grab
3. Научиться захватывать не первый доступный кадр, а десятый в v4l2grab и gStreamer

11.7. Выводы

В этой лекции мы провели краткий обзор возможностей, предоставляемых MeeGo API для работы с мультимедиа, сообщениями и аппаратными датчиками. Кроме того, кратко были рассмотрены основные способы коммуникации между приложениями и пример обработки видеокадров.

11.8. Контрольные вопросы

1) Что является типом элемента в конвейере gStreamer:

Варианты:

1. Модуль
2. Фильтр
3. Драйвер
4. Слот

2) Что представляет собой видеокамера в конвейере gStreamer:
   1. Источник
   2. Перехватчик
   3. Кодек
   4. Виртуальное устройство
3) Приведите пример фиктивного приемника в gStreamer:
   1. v4l2src
   2. fakesink
   3. pipeline
   4. ximagesink
4) Какие сигналы использует gStreamer для асинхронного взаимодействия:
   1. свои собственные
   2. Qt
   3. GTK
5) Какой тип имеет элемент «ffmpegcolorspace»:
   1. Источник
   2. Фильтр
   3. ffmpeg
   4. Приемник
6) Зачем в нашем приложении применяется конвертация видео:

1. Для демонстрации возможностей gStreamer
2. Для согласования возможностей камеры и экрана
3. Чтобы явно задать формат видео перед сжатием в jpeg
4. Без конвертации невозможна работа конвейера

7) Почему при сборке gstbrab используется pkg-config:
   1. Потому что gStreamer – пакет MeeGo
   2. Из-за невозможности использовать Makefile при разработке на целевой платформе
   3. Подключаемые при сборке библиотеки требуют большое количество параметров gcc

Список литературы

1. MeeGo Core API (http://apidocs.meego.com)
2. MeeGo Core API — Multimedia (http://apidocs.meego.com/1.1/core/html/qtmobility/multimedia.html)
3. The MeeGo Multimedia Stack (http://goo.gl/GCMdv)
4. MeeGo Media Framework (http://goo.gl/IwtMs)
5. MeeGo Core API — Sensors (http://apidocs.meego.com/1.1/core/html/qtmobility/sensors-api.html)
6. MeeGo Core API — Local Sockets (http://apidocs.meego.com/1.1/core/html/qt4/qtnetwork.html)
7. Сокеты (http://masters.donntu.edu.ua/2005/fvti/lukyanov/library/ipc/socket.html)
8. MeeGo Core API — D-BUS (http://apidocs.meego.com/1.1/core/html/qt4/qtdbus.html)
9. Посредник D-BUS (http://citkit.ru/articles/243/)
10. wikipedia — dbus (http://ru.wikipedia.org/wiki/D-Bus)
11. wikipedia — gstreamer (http://ru.wikipedia.org/wiki/GStreamer)
12. MeeGo wiki (http://wiki.meego.com)
13. Hendy's Qt Mobility Development (http://qt-mobility.blogspot.com/)
14. Qt 4.7.1 и Qt Mobility 1.1.0 / Qt Software / Хабрахабр (http://habrahabr.ru/blogs/qt_software/107959/)
15. Проект Video4Linux. http://linux.bytesex.org/v4l2/
16. Проект gStreamer. http://www.gstreamer.net/
17. Проект GTK. http://www.gtk.org/

Раздел 3. Разработка приложений

12. Разработка приложения для БПЛА на платформе MeeGo

Разработка приложения для БПЛА по захвату кадров от видеокамеры и передаче их через Интернет в ЦОД.

12.1. Проект «Мультиагентная система для БПЛА»

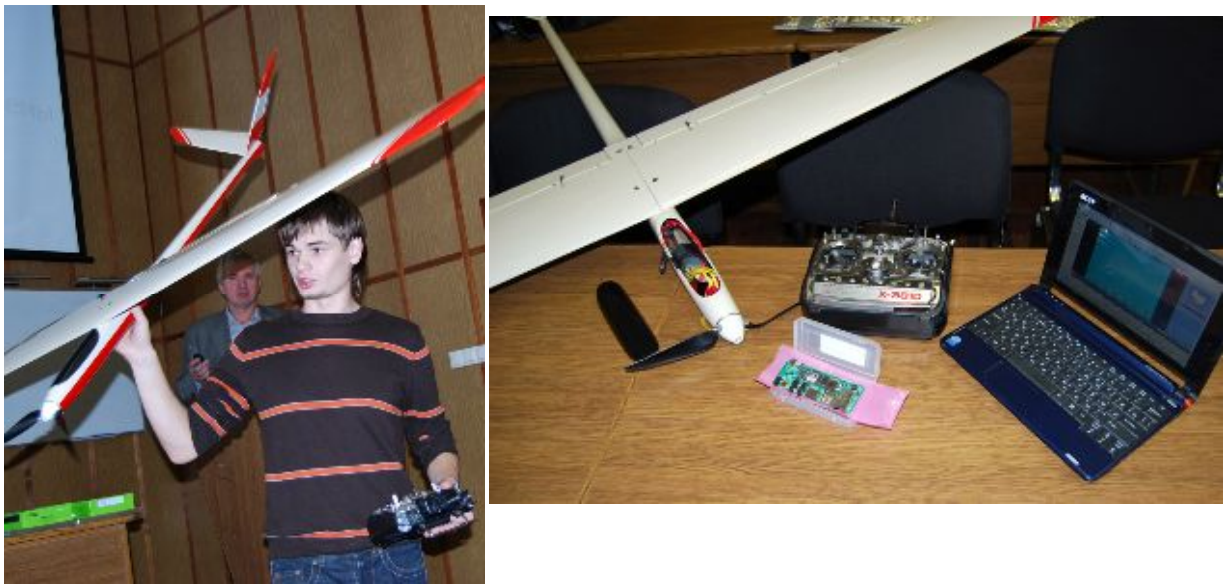


Рис. 12.1.1.

В нашем проекте, представленном в октябре 2010 года на "Петербургском международном инновационном форуме 2010", используется легкий планер PAPRIKA (см. Рис. 12.1.1) – размах крыльев 2 м, вес 2 кг, полезная нагрузка 500 г, скорость 40-60 км/ч, дальность 200 км, стоимость 25 тыс. руб. – с бесколлекторным электродвигателем, миниатюрной видеокамерой и бортовым микрокомпьютером Gumstix Overo Air размером 17 мм x 58 мм x 4,2 мм, который работает под управлением операционная система Linux и построен на основе процессора ARM Cortex-A8 с тактовой частотой 600 Mhz, имеет 256 MB памяти RAM и 256 MB встроенной памяти NAND Flash (Рис. 12.1.1). Связь между микрокомпьютерами разных БПЛА осуществляется за счет встроенного радиоприемника с частотой 2,4 GHz и протоколом общения 802.11 n (Wi-Fi), в котором применяется технология, связывающая два ближайших канала в один. Таким образом микрокомпьютеры в БПЛА смогут одновременно принимать и отправлять информацию друг другу. Связь с базовой станцией осуществляется за счет отдельного радиоканала или через GPRS по GSM модему.

За счёт небольшого веса БПЛА взлёт осуществляется с рук человека (в стандарте предполагается взлёт с катапульты). Посадка – либо за счёт встроенного парашюта, либо за счёт «перехвата» на ручное управление. Наличие «на борту» полнофункционального микрокомпьютера позволяет использовать БПЛА не просто в режиме «удаленной видеокамеры», которая передает видеопотоки в зоне прямой радиосвязи, а как интеллектуального мобильного агента с функциями автопилота. Такой агент способен существенно переработать исходную информацию, подготовить ее для обработки и самостоятельно автономно передать ее суперкомпьютеру по разным каналам связи – в том числе и через Интернет). Реальное ускорение в обработке данных позволит перевести в практическую

плоскость и организацию автоматической обратной связи от суперкомпьютера к БПЛА с указанием возможных корректировок планов полета и наблюдений. Бортовые микрокомпьютеры со связью позволяют организовать и автономное взаимодействие внутри группы БПЛА для возможности перераспределения или уточнения заданий, обмена данными и т. п.

Например, при использовании группы БПЛА для геологоразведки на определенной территории работа системы организуется следующим образом:

1. Выбирается тип задачи (в выбранном примере – геологоразведка).
2. В зависимости от площади исследуемой территории и количества БПЛА в группе, их характеристик, территория разделяется на участки, и формируются отдельные задачи для каждого члена группы.
3. В микрокомпьютер каждого БПЛА группы записывается глобальная задача (параметры исследуемой территории и т. п.) и отдельная задача этого самолета-агента.
4. Каждый агент приступает к выполнению поставленной ему задачи.
5. Когда в зону Wi-Fi видимости одного БПЛА из группы попадает другой, при «общении» происходит передача между агентами накопленной информации и при необходимости взаимное уточнение отдельных задач. (Таким образом по ходу выполнения частной задачи, все агенты накапливают информацию о ходе решении общей задачи группы, а также локально принимают решения о корректировке своих частных задач для более эффективного выполнения общей. Например, группа БПЛА летит по определённому маршруту, все БПЛА летят на разной высоте, при обмене информацией между членами группы выясняется наилучшая высота полёта по энергозатратам, по ходу выполнения задания вся группа оперативно перестраивается на эту высоту.)
6. Базовые наземные станции, обеспечивая связь с центром обработки данных (ЦОД), принимают/передают информацию от БПЛА, находящихся в их зоне видимости или поддерживающих связь через Интернет. Так как в процессе общения между БПЛА информация о выполнении общей задачи накапливается во всех микрокомпьютерах группы, то данные даже от тех самолетов, которые редко выходят на связь, все равно попадает в ЦОД.
7. Полученная в ЦОДе информация обрабатывается и визуализируется для заказчика (выдается карта с нанесенными исследуемыми характеристиками).
8. Наличие обратной связи с мобильными агентами (БПЛА) позволяет оперативно формировать из ЦОДа инструкции по корректировке их заданий.

Рассмотрим далее простейшую задачу для одиночного БПЛА – видеонаблюдение, которое будет заключаться в передаче серии фотоснимков с камеры, установленной на БПЛА, в ЦОД через GSM/GPRS канал.

12.2. Лабораторная работа № 10 «Разработка ПО бортового микрокомпьютера БПЛА: Передача файлов по сети»

12.2.1. Цель лабораторной работы

Демонстрация процесса разработки практического приложения.

12.2.2. Инструкция по выполнению лабораторной работы

Задача видеонаблюдения естественным образом распадается на составные части. Во-первых, нам нужно уметь подключаться к Интернету 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.

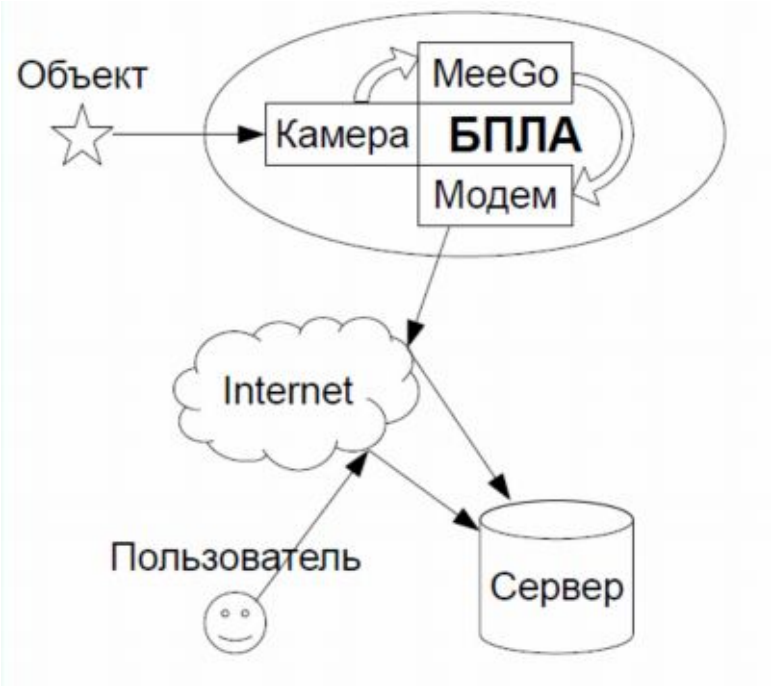


Рис. 12.2.1.

Будем считать, что бортовой компьютер БПЛА и ЦОД работают под управлением ОС MeeGo. Но также мы будем учитывать возможность замены ОС другим Linux, особенно на бортовом компьютере. Драйвера камеры интегрированы в ядро MeeGo и она доступна как устройство /dev/video0, это верно практически для любой современной USB вебкамеры. Драйвера модема также интегрированы в ядро MeeGo и он доступен как обычное символьное устройство, например, /dev/ttyUSB0, мы использовали GSM USB модем MF100 фирмы ZTE. В случае использования модема, предоставляемого провайдером сотовой связи, нужно иметь в виду, что провайдеры часто привязывают модем к своей сим-карте или программному обеспечению – это может затруднить использование модема.

Рассмотрим подключение к Интернету, используя USB GSM модем. Стандартный модуль ядра usbserial превращает современный USB-модем в символьное устройство, и мы можем работать с модемом так же, как работали с его аналогом на проводной телефонной линии 15 лет назад. Система АТ-команд для управления модемом также практически не изменилась. Мы можем пойти разными путями:

1. Использовать непосредственную настройку демона pppd, отвечающего за установление point-to-point соединения. Он наиболее широко распространен в мире Linux, но наименее удобен в настройке.
2. Использовать программу дозвона, такую как wvdial. Ее проще настраивать, но она может не входить в стандартную поставку Linux, особенно для бортового компьютера, и надо будет дополнительно компилировать ее.
3. Наконец, в MeeGo есть библиотека oFono (см. лекцию 7), одна из задач которой – установка GPRS-соединений. Список поддерживаемых модемов приведен в http://ofono.org/wiki/ofono-supported-modems. Главный недостаток – oFono достаточно массивная, сложная и сырая, с ее переносом на другие платформы могут быть проблемы.

   Далее описан первый вариант с использованием демона pppd.

   Установите его, выполнив

```
yum install ppp
```

(как обычно, для установки и настройки системных утилит нужны права администратора). После успешной установки необходимо создать конфигурационный файл /etc/ppp/peers/modem со следующим содержимым:

```
/dev/ttyUSB2 460800
connect '/usr/sbin/chat -v -f /etc/ppp/chat-modem'
defaultroute
replacedefaultroute
noauth
init 'echo -e "nameserver 8.8.8.8" > /etc/resolv.conf'
```

/dev/ttyUSB2 – имя устройства, через которое доступен модем. Один физический модем может создавать несколько устройств, не каждое способно работать в качестве модема, нужно будет попробовать каждое устройство и выбрать то, которое нормально работает. Опции defaultroute и replacedefaultroute указывают pppd на необходимость задать устанавливаемое соединение как маршрут по умолчанию. Последняя строка задает универсальный адрес сервера доменных имен (DNS), который будет работать с любым провайдером.

Также необходимо создать конфигурационный файл /etc/ppp/chat-modem для утилиты chat:

```
TIMEOUT 5
ECHO ON
ABORT '\nBUSY\r'
ABORT '\nERROR\r'
ABORT '\nNO ANSWER\r'
ABORT '\nNO CARRIER\r'
ABORT '\nNO DIALTONE\r'
ABORT '\nRINGING\r\n\r\nRINGING\r'
'' \rAT
TIMEOUT 12
OK ATZ
OK ATH
OK 'ATQ0 V1 E1 S0=0 &C1 &D2'
OK 'AT+CGDCONT=1,"IP","your-providers-apn.net"'
OK AT+CSQ
OK ATDT*99***1#
CONNECT ''
```

Параметры `your-providers-apn.net` (имя точки доступа) и `*99***1#` (номер для дозвона) зависят от провайдера услуг связи. Установка соединения осуществляется командой

```
pppd call modem
```

Его проверку можно выполнить в три этапа: во-первых, комадной

```
ifconfig
```

проверить появление сетевого интерфейса *ppp0*; во-вторых, командой

```
ping 8.8.8.8
```

проверить маршрутизацию пакетов; в-третьих, командой

```
ping google.com
```

проверить работоспособность DNS. Завершить соединение можно командой

```
kill `cat /var/run/ppp0.pid`
```

или прервав процесс pppd как-либо иначе.

Задаче захвата кадра камеры и сохранения его в jpg-файле была детально рассмотрена в лабораторной работе № 11.

Перейдем к третьей подзадаче – передаче файла на сервер и обеспечение к нему последующего доступа. При решении этого вопроса наиболее важно выбрать стандартный, унифицированный протокол, и первое, что приходит в голову – использовать HTTP. Соответственно, на ЦОД можно применить самое популярное в этой области решение – веб-сервер Apache со скриптовым движком PHP. Использование этой связки очень широко освещено в различной документации, для наших целей будет достаточно базовой установки

```
yum install httpd php
```

после чего надо дать PHP-скриптам право записывать файлы в директорию веб-сервера

```
chmod a+w /var/www/html
```

Если на ЦОД установлена иная ОС, то способ установки веб-сервера и его директория могут отличаться. Также мы не обсуждаем здесь безопасность этого решения.

В директории веб-сервера разместим файл upload.php:

```
<?php
if (@$_REQUEST['submitname']=="OK") {
  $target_path = "./";
  $target_path = $target_path.basename($_FILES['filename']['name']);
  if(move_uploaded_file($_FILES['filename']['tmp_name'], $target_path)) {
    echo "The file ".basename($_FILES['filename']['name']).
                                                    " has been uploaded\n";
  } else{
    echo "There was an error uploading the file, please try again!\n";
  }
  exit;
}
?>

<form enctype="multipart/form-data" action="upload.php" method="POST">
Choose a file to upload: <input name="filename" type="file" />
<input type="submit" name=submitname value="OK" />
</form>
```

Первая его часть содержит обработчик на PHP, который извлекает файлы из специального HTTP-запроса и размещает их в директории веб-сервера. Вторая часть – форма HTML для формирования этих запросов через браузер. Форма для наших целей не нужна, но ее можно использовать для тестирования первой части.

Для бортового компьютера нужна клиентская часть HTTP, снова можно выбрать из нескольких вариантов:

1. Можно воспользоваться консольным HTTP-клиентом, таким как cUrl и wget. Они умеют отправлять HTTP-запросы, но не занимаются отображением полученного HTML, JS-движками и т.п. Их удобно использовать для простых задач в скриптовых языках.
2. Также существуют аналогичные по функциональности библиотеки, например, libcUrl. Их можно сделать частью своего приложения.
3. Наконец, можно использовать Qt Network, в котором есть поддержка протокола HTTP.

Мы выберем самый простой вариант 1. Устанавливаем cUrl в MeeGo

```
yum install curl
```

хотя он уже должен быть установлен по умолчанию. Проверим, что cUrl и веб-сервер взаимодействуют нормально:

```
curl -F submitname=OK -F filename=@image.jpg http://server.com/upload.php
```

Здесь server.com – имя или IP-адрес сервера, где был настроен веб-сервер, а image.jpg – имя файла, который мы получили от камеры. В браузере оцениваем результат по адресу http://server.com/image.jpg.

Теперь надо объединить подзадачи в итоговое приложение. Мы не будем включать в итоговое приложение подключение к Интернет по модему, при желании это легко сделать по аналогии. У нас опять есть несколько вариантов:

5. Программа на скриптовом языке, например Perl или даже bash. Это наиболее быстрое и простое решение, но с плохими перспективами в плане развития.
6. Обычная linux-программа на C – компромисс между предыдущим и следующим вариантами.
7. QT-приложение с графическим интерфейсом даст использовать потенциал MeeGo на полную мощность.

Выбираем вариант 3. Проект состоит из трех файлов – camserver.pro, camserver.h и camserver.cpp. Мы унаследуем класс CamServerWidget от QWidget и добавим в него два слота – `grab()` и `start()` и таймер `grabTimer`. Пользовательский интерфейс будет состоять всего из трех кнопок *start*, *stop* и *quit* и мы добавим их прямо в исходном коде, в конструкторе класса, без применения специальных инструментов. Там же свяжем слоты и сигналы: сигнал `clicked()` кнопки quit свяжем со слотом `quit()`

всего приложения, чтобы завершать его работу. Сигнал кнопки stop свяжем со слотом `stop()` таймера, чтобы останавливать работу таймера. Сигнал кнопки start свяжем со слотом `start()` нашего класса CamServerWidget. Сигнал `timeout()` таймера свяжем со слотом `grab()` нашего класса, в нем будет вся функциональность.

Слот `start()` у нашего класса нужен только для того, чтобы первоначальный запуск таймера происходил сразу же, через 0 миллисекунд, а не через 6000 миллисекунд (6 секунд), которые задаются для каждого последующего запуска таймера. При этом установку нулевого таймаута нельзя перенести из слота в конструктор, потому что в этому случае нулевой таймаут сработает только при первом нажатии кнопки *start*, а если мы остановим процесс и запустим его заново, то таймаут уже окажется ненулевой.

Слот `grab()` вызывается таймером с заданной периодичностью и сначала вызывает приложение для захвата кадра, а затем cUrl для отправки файла на сервер. Для обеих операций установлены таймауты, чтобы суммарное время выполнения не превышало интервал таймера.

Собирать это приложение мы будем так же прямо в MeeGo. Установим зависимости

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

Собираем с помощью

```
qmake; make
```

Перед запуском убеждаемся, что приложение v4l2grab находится в той же директории, камера подключена, веб-сервер доступен по сети. Запускаем приложение и нажимаем кнопку *Start*. В браузере проверяем, что картинка на нашем сервере обновляется каждые 6 секунд. БПЛА можно отправлять на задание.

Конечно, созданное приложение является всего лишь прототипом. Однако и с его помощью можно сделать многое – уточнить требование к приложению, проверить совместимость аппаратуры, оценить требуемые ресурсы, быстро получить тестовое окружение, провести убедительную демонстрацию. Наконец, прототип можно усовершенствовать и постепенно превратить в полноценное приложение.

12.2.3. Задания для самостоятельной работы

1. Интегрировать libcUrl в Qt-приложение
2. Отказаться от cUrl/libcUrl и использовать Qt Network

12.3. Выводы

В этой лекции мы начали рассматривать пример разработки практического приложения по захвату видеокадров, сохранению их п передаче через Интернет в ЦОБ. В следующей лабораторной работе разработка примера будет продолжена.

12.4. Контрольные вопросы

1) Где расположен драйвер камеры в MeeGo:
   1. BIOS
   2. Ядро Linux
   3. XWindows
   4. QT Multimedia Framework
2) Как именуется камера среди устройств MeeGo:
   1. /dev/video0
   2. /dev/camera
   3. /var/webcams/1
   4. Имя камеры устанавливается ее производителем
3) Каким образом доступен модем под Linux:
   1. Как блочное устройство

2. Как символьное устройство
3. Как модуль ядра
4. Как интерфейс DBus

4) Может ли поставщик услуг связи ограничить использование предоставляемого им модема:
   1. Нет, не может
   2. Может только программно
   3. Может только аппаратно
   4. Может и программно, и аппаратно
5) Какая ОС использует тот же менеджер пакетов, что и MeeGo:
   1. Windows
   2. Ubuntu
   3. Fedora
   4. Debian
6) В каком формате наиболее удобно сохранение видеокадра в файл:
   1. JPG
   2. GIF
   3. XML
   4. HTML
7) Каким образом современный пользовательский GSM-модем подключается к компьютеру:
   1. По шине ISA
   2. Через порт UART
   3. По шине SPI
   4. Через порт USB
8) Как традиционно называется набор команд для управления модемом:
   1. AT команды
   2. AT&T команды
   3. PPP команды
   4. PPPD команды
9) PPPD – это:
   1. Сервис
   2. Демон
   3. Интерфейс
   4. Сервер
   5. Библиотека
10) Wvdial – это:
   1. Модуль ядра для превращения USB-устройства в символьное
   2. Компонент oFono для поддержки GPRS-соединений
   3. Программа для дозвона
11) Какой инструмент для установления соединений через GSM-модем имеет наименьшее количество зависимостей:
   1. PPPD
   2. wvdial
   3. oFono
12) Какой способ установления соединений через GSM-модем предпочтителен в MeeGo:
   1. PPPD
   2. wvdial
   3. oFono

13) Установка PPPD в MeeGo осуществляется командой:
    1. yum install pppd
    2. apt-get pppd
    3. yum install ppp
    4. ppp install
14) Для чего предназначены опции defaultroute и replacedefaultroute в файле конфигурации PPPD:
    1. Обнаружение модема в системе
    2. Создание сетевого интерфейса
    3. Настройка маршрутизации пакетов
    4. Настройка DNS
    5. Управление HTTP-прокси
15) Какая утилита входит в пакет и используется PPPD в процессе работы:
    1. telnet
    2. killall
    3. top
    4. chat
16) Как соотносятся конфигурационный файл утилиты chat и AT-команды:
    1. Файл не содержит AT-команд
    2. Файл включает в себя в том числе и AT-команды
    3. Файл состоит исключительно из AT-команд
17) Что выводит на экран утилита ifconfig:
    1. Список установленных TCP-соединений
    2. Дамп IP-пакетов
    3. Список активных сетевых интерфейсов
    4. Адреса серверов DNS
18) Что позволяет проверить успешное выполнение команды «ping 8.8.8.8»:
    1. Наличие USB-модема
    2. Наличие сетевого интерфейса 8.8.8.8
    3. Наличие подключения к Интернету и корректной маршрутизации
    4. Наличие подключения к Интернету и корректной работы DNS
19) Интерфейс v4l2 следует использовать для:
    1. Обеспечения совместимости с Windows
    2. Написания видеоприложений на Java
    3. Использования механизма слотов и сигналов
    4. Работы непосредственно с ядром Linux
20) Какая из перечисленных библиотек отсутствует в MeeGo:
    1. Qt Multimedia Framework
    2. Qt Phonon
    3. gStreamer
21) Каких пакетов, помимо предустановленных, достаточно для сборки v4l2grab непосредственно в MeeGo:
    1. gcc и libjpeg-devel
    2. gcc и make
    3. gcc-c++ и qt-devel
22) Какой командой следует осуществлять сборку v4l2grab:
    1. make v4l2grab
    2. gcc -c v4l2grab.c –DJPEG

3. gcc v4l2grab.c -o v4l2grab -ljpeg

23) Какой протокол используется для отправки файла на веб-сервер:
    1. SSH
    2. XMPP
    3. HTTP
    4. SIP
24) Какая веб-платформа может быть использована в Linux:
    1. Microsoft IIS + ASP
    2. Apache + PHP
    3. Nginx + memcached
25) Что делает команда chmod a+w /var/www/html:
    1. Дает всем пользователям права на запись в директорию веб-сервера
    2. Позволяет веб-серверу запускать PHP-скрипты от имени администратора
    3. Форматирует директорию веб-сервера для последующего использования
26) Какая из перечисленных утилит не является HTTP-клиентом:
    1. talk
    2. wget
    3. curl
27) В какой библиотеке Qt есть поддержка протокола HTTP:
    1. Qt TCP
    2. Qt Socket
    3. Qt Network
28) Из каких двух частей состоит скрипт upload.php:
    1. Форма HTML и описание стиля
    2. Форма HTML и PHP-обработчик
    3. AJAX-обработчик
    4. Два PHP-обработчика для приема и отправки файлов
29) Может ли скрипт upload.php принять несколько файлов в одном HTTP-запросе:
    1. Может
    2. Может, если файлы передаются внутри самораспаковывающего архива
    3. Не может
30) Как правильно отправить файл image.jpg скрипту upload.php с помощью curl:
    1. curl -F submitname=OK http://server.com/upload.php < image.jpg
    2. curl -F submitname=OK -F filename=@image.jpg http://server.com/upload.php
    3. curl http://server.com/upload.php?submitname=OK&filename=image.jpg
    4. С помощью curl файл отправить невозможно
31) Какой язык программирования нельзя отнести к скриптовым:
    1. bash
    2. Perl
    3. C
    4. PHP
32) От какого класса унаследован класс приложения camserver:
    1. QWidget
    2. QApplication
    3. QMain
33) Какой слот таймера используется для его остановки:
    1. timeout()
    2. stop()

3. disable()

34) Какой слот приложения используется для завершения работы:
quit()
exitapp()
stop()

35) Где задается внешний вид интерфейса пользователя приложения camserver:
1. В файле проекта
2. Во внешнем файле ресурсов
3. В исходном тексте

36) В каких единицах задается интервал таймера:
1. В наносекундах
2. В микросекундах
3. В миллисекундах
4. В секундах

37) С какой целью установлены таймауты для вызовов внешних программ из camserver:
1. Чтобы ускорить работу внешних программ
2. Это необходимо для корректной работы класса QProcess
3. Чтобы время работы внешних программ не превышало интервал таймера

38) Что произойдет при вызове grabTimer->start(0):
1. Сигнал timeout() будет испущен немедленно
2. Сигнал timeout() будет испущен при срабатывании деструктора grabTimer
3. Произойдет ошибка

39) Каких пакетов, помимо предустановленных, достаточно для сборки Qt-приложения camserver непосредственно в MeeGo:
1. gcc и qt-timer-devel
2. make, gcc-c++ и qt-devel
3. make и gcc-qt

40) С помощью чего можно убедиться в правильности работы приложения camserver в целом:
1. С помощью отладчика
2. С помощью Qt Linguist
3. С помощью браузера

Список литературы

1. Проект cUrl и libcUrl. http://curl.haxx.se/
2. Проект Apache http://www.apache.org/
3. Проект PHP http://www.php.net/
4. Введение в PPPD. http://en.wikipedia.org/wiki/Point-to-Point_Protocol_daemon

13. Реализация качества разработки программных приложений

Качество и эффективность программного обеспечения. Некоторые аспекты стандартизации процесса разработки программного обеспечения. Отечественные стандарты оценки качества программных продуктов. Стандарты ISO.

13.1. Качество и эффективность программного обеспечения

Качество и эффективность программного обеспечения (программных систем, средств, инструментов, приложений), работающего в государственных и бизнес-структурах, служащего основой всемирной сети и разнообразных информационных систем, являются критически важными факторами. Сегодня деятельность многих организаций, предприятий и, особенно, высокотехнологичных компаний напрямую зависит от качественной обработки информации соответствующими компьютерными системами. Разработка программного обеспечения является сложным и многоаспектным процессом, в котором участвует большое количество специалистов различных профилей и различной квалификации. Кроме того, в процессе разработки ПО сплетаются множество технических, технологических и управленческих проблем. От их адекватного разрешения зависит успех проекта, качество произведенного продукта и, как следствие, рейтинг компании и конкурентоспособность на рынке программных средств.

Некачественное программное обеспечение может привести к серьезным потерям и затратам на восстановление утерянных данных и программной системы в целом. Даже если и удается избежать чрезвычайных происшествий, это наносит ущерб эффективности работы, что особенно негативно сказывается на коммерческих предприятиях, поскольку снижает их конкурентоспособность. Предприятия наиболее уязвимы в период внедрения и использования новых программных продуктов и проведения реинжиниринга процессов. Именно в это время ущерб от некачественного программного обеспечения наиболее вероятен [1].

По известным данным исследования Тома де Марко (США):

- 15% всех программных проектов так и не достигли своего завершения;
- превышение стоимости проектов на 100 – 200% является обычным явлением;
- превышение стоимости на 30% в программной индустрии считается настоящим успехом.

Эти исследования проведены в конце прошлого века, однако серьезность проблем к настоящему времени не только не уменьшилась, но по ряду проблем существенно возросла – особенно в России.

Проблемы, выявленные в ходе 151 оценок различных программных проектов в США:

- помехи со стороны правительства – 10%;
- технологические факторы – 12%;
- внешние поступления – 12%;
- оборудование – 15%;
- недостаточность контроля – 35%;
- недостаточность/недостатки планирования производственных процессов – 55%.

Выдержка из статьи «Software’s Chronic Crisis», появившейся в журнале «Scientific American», через 12 лет после опубликования этого исследования (автор W. Wayt Gibbs, сентябрь 1994): «Исследования показали, что на каждые 6 крупных систем программного обеспечения, запущенных в действие, приходится 2 (т. е. 30-35%) таких, разработка которых была прекращена из-за

невозможности добиться удовлетворительного функционирования. Средний проект разработки программного обеспечения затягивается на половину первоначально запланированного срока, крупные проекты и того хуже. Три четверти всех больших систем либо выполняют не все функции, которые на них возлагались, либо не используются вовсе».

По данным Департамента по Торговле и Промышленности Великобритании (DTI) при внедрении проектов информационных технологий на предприятиях потери из-за некачественного программного обеспечения составляют в среднем около 20% от общего объема потерь. По разным оценкам аналогичный показатель для России достигает величины от 30 до 50%.

Может показаться, что управлять количеством ошибок в коде невозможно и их число зависит только от способностей конкретного разработчика. Это не совсем так. В конце 80-х – начале 90-х годов в программной индустрии был проведен ряд серьезных исследований эффективности труда программистов. Вот данные американского Института программной инженерии (Software Engineering Institute – SEI). Профессиональные программисты со стажем 10 и более лет на 1000 строк кода (СК – строка исходного текста, которая содержит программную инструкцию) допускают в среднем 131,3 ошибки. Следовательно, большая система размером миллион СК может содержать 100 тысяч ошибок! Если транслятор имеет развитую систему предупреждений, то до 50% из них выявляется на этапе компиляции.

На этапе тестирования отдельных модулей (типичный программный модуль по определению SEI содержит от 50 СК до 5 тысяч СК) обнаруживается половина оставшихся ошибок. Получается, что перед этапами внедрения и комплексного тестирования в продукте еще скрывается 25 тыс. ошибок. Для их устранения на заключительных этапах тратится от 10 до 40 человеко-часов на ошибку, т. е. *на доведение продукта до идеального состояния потребуется 125 человеко-лет работы!* Это, конечно, сильное преувеличение, но суть именно такова – на исправление ошибок за весь период сопровождения продукта суммарные затраты времени (в человеко-месяцах) могут быть весьма значительными!

Можно привести другой пример. Типичный небольшой продукт имеет объем до 50 тысяч СК. Пусть его создают 5 программистов, делая при этом 100 ошибок на тысячу СК. 50% ошибок выявляется на этапе компиляции с незначительными расходами времени, устранение ошибок на этапе тестирования занимает 90% времени. Стоимость устранения одной ошибки в готовом продукте по данным Microsoft оценивается в 4 тыс. долл. (по данным IBM устранение ошибок в ее продуктах, выпущенных в эксплуатацию, обходится от 2-х до 20 тысяч долларов на ошибку) [2].

Корень зла кроется в неправильных акцентах при управлении качеством ПО!

Корреляция между числом ошибок, обнаруженных при тестировании отдельных модулей, и числом ошибок, найденных пользователями в готовом продукте, равна 0,91. Отсюда вывод – если на тестирование поступит некачественный продукт, он таким и будет выпущен в продажу! Таким образом, организация производства качественного программного обеспечения является *ключевой проблемой управления процессом разработки* ПО, которой в индустриально развитых странах стали активно заниматься примерно с середины 60-х годов. К числу бесспорных достижений теории менеджмента качества относятся современное понятие качества и его смысловое наполнение, а также известные модели систем качества серии ISO 9000:2000 (ISO 9001, ISO 9002 и ISO 9003), построенные на базе процессного подхода.

Согласно современному определению из международного стандарта, качество программного обеспечения, как и любого другого продукта, – это его соответствие потребностям потребителя (пользователя). Самый верный путь повысить качество ПО – постоянно улучшить и совершенствовать процесс создания и сопровождения продукта в масштабах компании, включающий процессы взаимодействия с внешней средой!

Анализ нормативных документов показывает, что к числу наиболее значимых показателей качества, основанных на обоснованных претензиях пользователей, можно отнести:

- неадекватность функционирования программного продукта;
- недостаточное взаимодействие продукта с другими программными, аппаратными и телекоммуникационными средствами;
- отказы программного продукта в процессе применения по назначению;

- замедленное время работы программного продукта и задержки представления им промежуточной и выходной информации;
- неполнота отражения информации;
- несоответствие хранимых данных и информации, вводимой оператором;
- потеря актуальности информации, циркулирующей в информационной системе;
- нарушения конфиденциальности информации;
- содержанию сопроводительной документации и справочной системе программного продукта.

Кроме таких, «первичных» данных о качестве, идущих непосредственно от потребителя, в роли показателей могут использоваться:

- число строк кода в стандартном модуле, количество выявленных ошибок на 1000 строк кода,
- вероятность появления специфических ошибок,
- параметры сложности программы,
- стоимость единицы кода,
- цена «человека-месяца»,
- статистические характеристики процессов (математические ожидания, дисперсии, корреляционные функции и т. д.) и другие оценочные параметры.

Отсюда следует «генеральная» задача разработчика ПО – на основе анализа взаимной корреляции ранжированных потребительских, технологических и технических характеристик создать систему целевых показателей (метрик), которая задает ориентиры разработки и критерии оценки ее качества. Такая задача трудна даже для искушенного разработчика, однако использование ранее наработанного опыта и знаний, накопленных в межпроектных базах данных и отраженных в корпоративных и международных стандартах, позволяют существенно уменьшить «размерность» задачи.

Выше было отмечено, что из-за отсутствия адекватных систем управления качеством во многих проектах (особенно крупных) временные и экономические показатели значительно превышаются по отношению к запланированным. При этом, однако, не гарантируется выполнение и технических требований. Ряд крупных IT-проектов, как в Европе, так и в Америке, не достигли заявленных результатов, будучи не в состоянии реализовать требуемые технические и технологические параметры, хотя отпущенные время и средства были значительно превышены.

Система управления качеством является частью системы управления организацией, которая ориентирована на достижение результатов, основанных на целях качества, удовлетворении нужд и ожиданий заказчиков. Цели в сфере качества дополняют основные (стратегические) цели организации. Различные части системы управления организации-разработчика могут быть объединены вместе с системой менеджмента качества в единую, унифицированную систему управления с общими элементами. Это способствует эффективному планированию, распределению ресурсов, установлению взаимодополняющих целей и реальной оценке эффективности предприятия. Инициативы внедрения систем качества в широких масштабах в Японии в начале 50-х годов, поддержанные правительственными программами, обеспечили быстрый рост её конкурентоспособности, и выход страны на лидирующие позиции в мире в ряде областей промышленности. Активное внедрение подходов к обеспечению качества в США и Европе началось в начале 60-х годов.

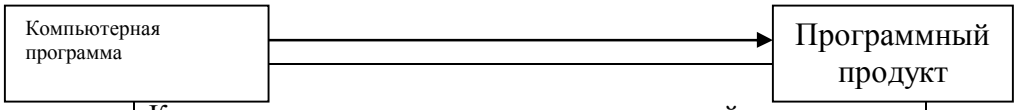
Если говорить о программировании, то идеи реализации качества на базе создания стандартного процесса разработки и сопровождения ПО пришли в эту область из промышленности в ответ на программный кризис конца 60-х годов [3]. Среди стандартов в области разработки систем качества, оценки качества процессов и уровней зрелости компаний, разрабатывающих программное обеспечение, а также совершенствования процессов в настоящее время наиболее популярными являются: ISO 9000 (версии 1994 и 2000 гг), ISO 12207, TickIT, SEI SW-CMM, Trilium, ISO 15504 (SPICE), CMMI. Знакомство студентов с этими широко применяемыми стандартами является важной частью курса.

13.2. Некоторые аспекты стандартизации процесса разработки программного обеспечения

Первая проблема, с которой приходится сталкиваться при попытке стандартизировать процесс разработки ПО, это интеллектуальная природа процесса. Несмотря на то, что программирование более или менее формализуется применением стандартных языков программирования и структурированием данных, сам процесс до сих пор является более искусством, чем ремеслом. Одну и ту же программу разные разработчики напишут по-разному, но очень немногие из полученных вариантов будут совершенными. Именно поэтому капитальный труд Д.Кнута назван *«Искусство программирования»*.

Компьютерная программа является материальным объектом, но она строится на абстрактных идеях и состоит из сложнейших виртуальных конструкций. Это в корне отличает ее от физических объектов, с которыми имеет дело обычное производство. Выше уже говорилось об этом, но виду важности обстоятельства повторим ещё раз – невозможно создать качественное ПО, не решив ряд типичных проблем [4]:

- команда разработчиков, как правило, состоит из творческих личностей и часто бывает очень трудно привести их к «общему знаменателю»;
- каждый программный продукт неизбежно содержит ошибки, отражающие квалификацию и индивидуальный стиль разработчика – разнообразие и нестандартность ошибок сильно усложняет процесс достижения стандартных целей в области качества;
- каждый успешный проект по-своему уникален и индивидуален, он подобен мозаике со сложным рисунком, и поэтому очень трудно выделить из него некий базовый процесс-клише, который можно было применить в дальнейших разработках;
- по сходной причине трудно поставить производство сложного и уникального ПО на поток, так как часто для его разработки требуется создание сопутствующего специфического программного «инструментария» для разработки, оптимизации и тестирования;
- одна из специфических проблем программирования состоит в том, что его продуктивность растет очень медленно, если растет вообще – по некоторым оценкам средний программист способен создать 10-50 строк операторов в день. Кроме того, эти оценки должны быть уменьшены для больших систем, так как увеличенная нагрузка требует увеличения затрат (в относительных единицах). Это обстоятельство резко отделяет программирование от другого вида деятельности, где поточное производство радикально уменьшает цену единицы продукта. Вся экономика строится на этом далеко не новом принципе – *но только не производство программного обеспечения!*



Концептуальное проектирование программной системы
Установление требований и ограничений
Спецификация системных и функциональных требований
Идентификация и документирование ключевых процессов
Архитектурное (логическое) проектирование системы
Построение базовых алгоритмов
Разработка стандартов на программирование
Кодирование программ
Первичное тестирование кодов
Поблочное и функциональное тестирование
Системная интеграция программного обеспечения
Интеграционные и эксплуатационные тесты
Документирование программного продукта
Сопровождение программного продукта

Рис. 13.2.1. Структура базового процесса для реализации жизненного цикла программного продукта.

В большинстве случаев справедливо заключить, что разработка ПО, особенно нестандартного, является в большой степени искусством, нежели ремеслом. Вследствие этого попытки привести процесс программирования в жесткие рамки системы качества в этой области являются более трудными, чем в других видах производственной деятельности.

На Рис. 13.2.1. показано содержание процесса, превращающего компьютерную программу в программный продукт. Совершенно очевидно, что эти *ключевые области* жизненного цикла программных средств обязательно должны быть «погружены» в среду организационных и вспомогательных процессов создания инфраструктуры, управления конфигурацией, распределения ответственности, производственного и административного контроля, внутреннего аудита, обучения персонала, регулирования взаимоотношений «поставщик-покупатель» и т. д.

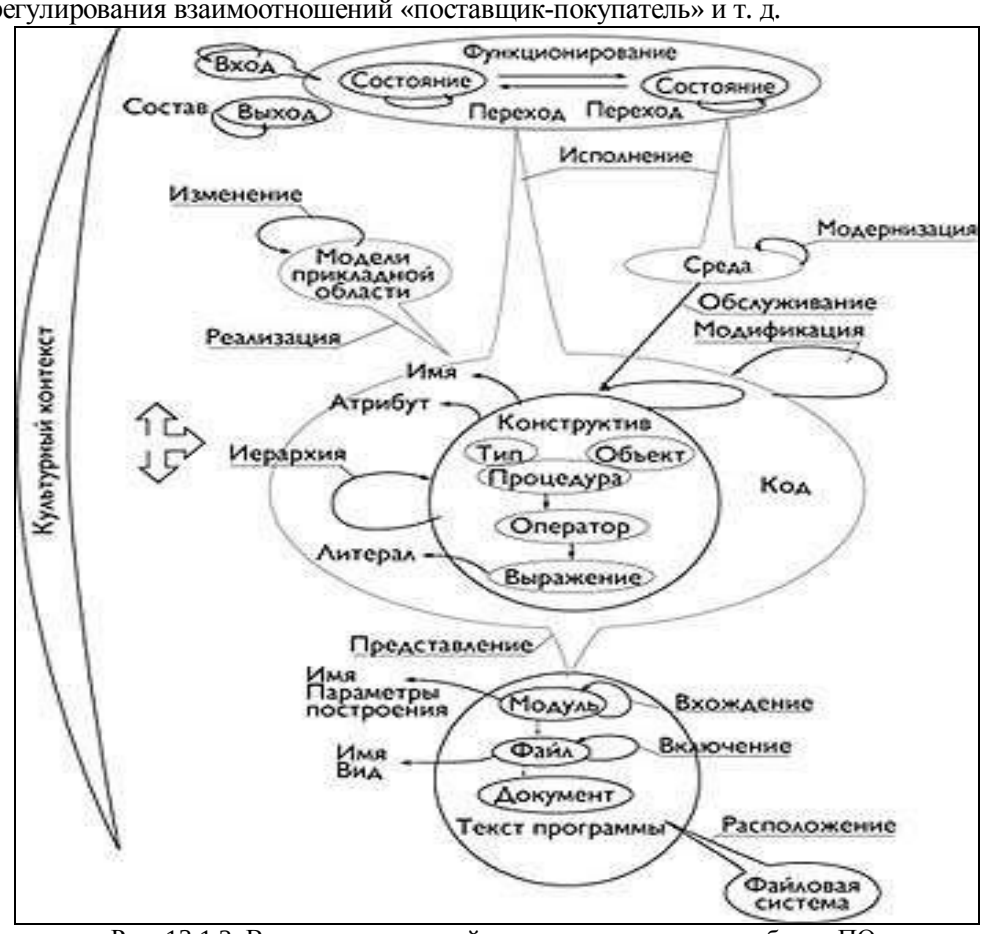


Рис. 13.1.2. Внутренняя понятийная среда процесса разработки ПО.

На Рис. 13.2.2 представлена концептуальная модель различных аспектов создания программного средства, отображающая внешнее и внутреннее информационное пространство, в котором действует разработчик ПО, а также наборы сущностей, атрибутов и состояний, образующих внутреннюю среду разработки. Процессы стандартизации и реализации качества не могут быть изолированы от общей системы знаний и опыта, образующей *культурный контекст* разработки ПО. Это достаточно широкая сфера, охватывающая практику решения задач в конкретной прикладной области, состояние программной среды и наличие необходимых инструментов, нормы языков программирования, применение новейших технологий и т. д.

Конкретные условия разработки, организация и структура коллектива разработчиков, принципы его работы – это еще один компонент контекста. Принимаемые решения, эффективность деятельности, конечный результат часто существенным образом зависят от того, организованы программисты в виде «хирургической бригады» Брукса или как группа равноправных соисполнителей. В любом случае, указанные аспекты составляют предметную область процесса реализации и сопровождения качества разработки программного обеспечения. Развитие этих идей,

постепенное их воплощение, накопление опыта компаний в создании эффективных методик и технических средств разработки и контроля качества процессов привели к тому, что в течение 20 постепенно сложилась определенная культура, появились традиции и стиль работы по созданию программных продуктов.

Наилучшие практики, взятые на «вооружение» ведущими компаниями, образовали так называемые «Тридцать четыре необходимые компетенции для управления программными проектами, стандартизации процесса разработки ПО и реализации его качества» [5]. Эти компетенции нашли отражение в большом количестве национальных и корпоративных стандартов разных стран, которые послужили основой для создания взаимосвязанной сети базовых и вспомогательных международных стандартов, широко применяемых в настоящее время (Рис. 13.2.3). Кроме общих международных стандартов на системы менеджмента качества, оценку и аудит процессов, существует ряд отечественных нормативных документов, конкретно посвященных качеству программных продуктов, упоминание которых интересно в ретроспективном плане. В Табл. 13.1 приведен список организаций, поддерживающих процесс разработки программного обеспечения на основе международных стандартов.

Таблица 13.1.

	Организация	Название	Основная цель деятельности	Продукт деятельности в области разработки ПО
1	Международная организация по стандартизации (International Organization for Standardization)	ISO	Разработка, актуализация, утверждение стандартов	Стандарты семейства ISO 9000, задающие модели систем качества и стандарт жизненного цикла разработки ПО ISO 122207
2	Институт инженерии по электротехнике и электронике (Institute of Electrical and Electronics Engineers)	IEEE	Разработка стандартов по инжинирингу в области электротехники и электроники	Собрание стандартов в области программного инжиниринга
3	Американский национальный институт стандартов (American National Standards Institute)	ANSI	Разработка американских национальных стандартов	Руководства, определяющие применение стандартов ISO и ISO 12207 к проектам в области программного инжиниринга
4	Национальный институт стандартов и технологий США (National Institute of Standards and Technology)	NIST	Разработка технологий, измерений и стандартов в американской промышленности	Стандарты и Национальная премия Малкольма Балдриджа за качество производства и продукта
5	Институт програм-много инжиниринга (Software Engineering Institute, USA)	SEI	Разработки в области программного инжиниринга	Модель зрелости возможностей компании, разрабатывающей ПО – модели CMM и CMMI
6	Американское общество обеспечения качества (American Society of Quality)	ASQ	Улучшение качества производимых продуктов	Систематизация знаний в области инжиниринга качественного ПО
7	Институт управления проектами (Project Management Institute)	PMI	Общие принципы управления проектами, в т.ч. проектами разработки ПО	Руководство «Основы знаний в области управления проектами» (PM BoK – Project Management Body of Knowledge)
8	Британский институт стандартов (British Standards Institute)	BSI	Разработка и актуализация британских национальных стандартов	Система британских национальных стандартов BS

13.3. Отечественные стандарты оценки качества программных продуктов

Рассмотрим ретроспективно некоторые российские стандарты оценки качества ПО. Это интересно потому, что это был, пожалуй, единственный случай в индустрии разработки ПО, когда российский ГОСТ был взят за основу при создании аналогичного стандарта ISO.

Прежде всего, это ГОСТ 28195-89 «Оценка качества программных средств. Общие положения», который устанавливает общие положения по оценке качества программных средств, поставляемых через фонды алгоритмов и программ, номенклатуру и применяемость показателей качества [6]. В стандарте отмечается, что оценка качества должна осуществляться на всех этапах жизненного цикла программных средств – при планировании показателей качества, его контроле на отдельных этапах разработки, в процессе производства, при проверке эффективности модификации на этапе сопровождения. Установлено, что оценку качества проводят специалисты организаций:

- разработчика – на этапах разработки;
- фондодержателя – на этапах приемки программного средства в Фонд алгоритмов и программ;
- испытательных и сертификационных центров – на этапах испытаний и внедрения; изготовителя – на этапах тиражирования;
- пользователя – на этапах внедрения, сопровождения и эксплуатации.

К основным задачам, решаемым при оценке качества программных средств, отнесены:

- планирование уровня качества;
- контроль значений показателей качества в процессе разработки и испытаний;
- эксплуатационный контроль заданного уровня качества;
- методическое руководство разработкой нормативно-технических документов по оценке качества.

С момента вступления в силу ГОСТ 28195-89 произошли существенные изменения во многих аспектах общественной жизни, в том числе значительно изменились экономико-правовые отношения в сферах разработки и эксплуатации программных средств. Например, в области коммерческих программных продуктов исчез фондодержатель, а разработчик и изготовитель обычно представляют собой одно и то же юридическое лицо. В рыночных условиях разработчик заинтересован в обеспечении качества своих продуктов в течение всего их жизненного цикла. Кроме того, изменился порядок сертификации продукции. Типичный порядок оценки качества программных продуктов, сложившийся в современных условиях, приведен в Табл. 13.2.

Таблица 13.2.

Порядок оценки качества программных продуктов

Заинтересованные стороны в оценке качества	Этапы жизненного цикла программного средства					
	Разработка	Испытания	Тиражирование	Внедрение	Сопровождение	Эксплуатация
Разработчик	Да	Да	Да	Да	Да	Да
Испытательные и сертификационные центры	-	Да	-	Да	-	Да
Пользователь	-	-	-	-	-	Да

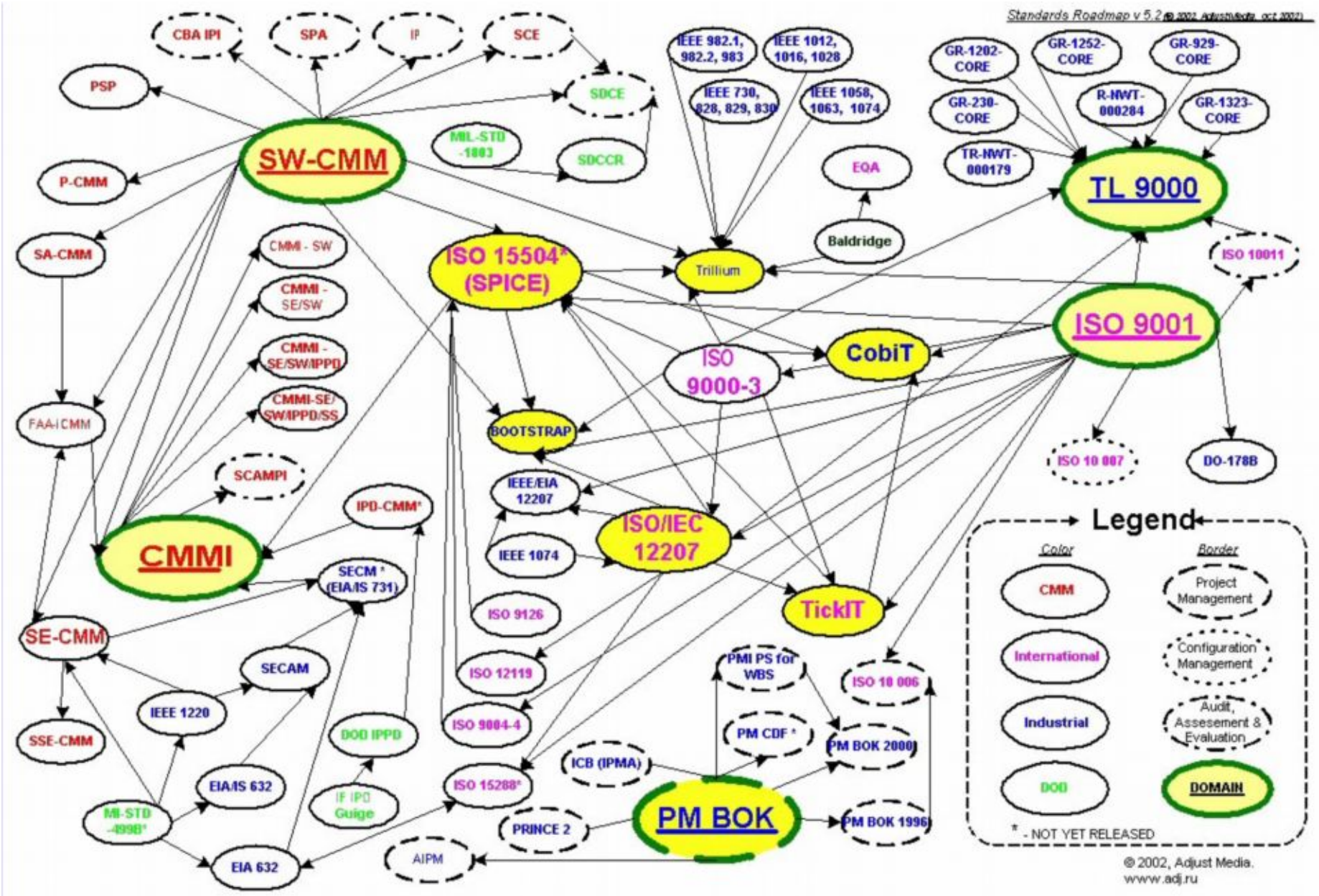


Рис. 13.2.3. Система взаимосвязанных корпоративных и международных стандартов.

Стандарт ГОСТ 28195-89 определяет иерархическую структуру, номенклатуру и содержание понятий качества программных средств. На верхнем уровне выделены шесть характеристик – *надежность, корректность, удобство применения, эффективность, универсальность и сопровождаемость*, которые детализируются на втором уровне 19 комплексными показателями. На третьем уровне дальнейшая детализация содержит более чем 200 оценочных элементов. Состав используемых показателей рекомендуется выбирать в зависимости от назначения, функций и этапов жизненного цикла программного средства.

В другом отечественном стандарте, имеющем отношение к рассматриваемой проблеме, ГОСТ 28806-90, установлены основные термины и определения понятий в области качества программных средств. В ГОСТ 28806-90 к общим характеристикам качества программного средства отнесены: *функциональность, надежность, удобство использования, эффективность и сопровождаемость*. В справочном приложении стандарта приведены примеры 20 подхарактеристик качества.

Сравнительный анализ стандартов показывает, что в различных нормативных документах номенклатура показателей качества программных средств заметно отличается друг от друга. Стандарты практически не содержат рекомендаций по выбору, применению и упорядочению необходимой совокупности показателей качества. Кроме того, для большинства показателей отсутствуют методики их оценки и сравнения с установленными требованиями (критериями). Практическое использование нормативных документов еще в большей степени осложняется тем, что ни один из них не содержит перечней факторов, влияющих на качество программных средств, и методик их количественной и качественной оценки.

13.4. Стандарты семейства ISO

В 1947 г. в Лондоне представители 25 стран решили создать международную организацию, основной задачей которой стала бы координация разработок и унификация международных стандартов. Новая организация получила название Международная организация по стандартизации (International Organization for Standardization –

ISO). На греческом языке *iso* означает *равный* – тем самым подчеркивалось, что все страны, входящие в эту организацию равны и каждая страна может на равных участвовать в формировании новых стандартов. В настоящее время ее членами являются более 100 стран. Главной причиной рождения стандартов явилось желание инициаторов создания этой организации устранить технические барьеры в торговле и в оценке качества продуктов, которые возникли вследствие того, что в разных странах для одних и тех же технологий и товаров действовали разнородные стандарты. Сегодня стандартами ISO «перекрыты» многие технологические отрасли – от программирования и телекоммуникаций до банковской и финансовой сферы. По мнению экспертов, в ближайшие годы стандарты ISO по-прежнему будут распространяться ударными темпами, что обусловлено целым рядом факторов:

- бурным прогрессом в области интеграции мировой торговли и промышленности, взаимопроникновением рынков. С технологической точки зрения свободная конкуренция немыслима без четко сформулированных, понятных и общепринятых стандартов. В современных условиях ни один рынок не может обойтись без компонентов, производимых на других рынках, а цифровая обработка данных требуется повсеместно;
- глобальным распространением компьютерных коммуникационных средств. Вычислительная индустрия – яркий пример технологии, существование которой абсолютно немыслимо без общепринятых стандартов. Полная совместимость открытых систем способствует развитию здоровой конкуренции среди производителей;
- потребностью в наличии общих стандартов в процессе становления новейших технологий. На ранних этапах их развития возникает необходимость в стандартизации терминологии и способах представления и хранения количественной информации.

По уставу ISO членом этой организации может стать "самая авторитетная в стране организация, занимающаяся выработкой стандартов". Таким образом, интересы какой-либо страны в ISO может представлять только одна организация. Помимо основных

членов, в ISO входят так называемые члены-корреспонденты (как правило, ими становятся относительно крупные «стандартообразующие» организации отдельных стран, однако еще недостаточно мощные, чтобы распространить свое влияние на стандартизацию технологий по всей стране). Члены-корреспонденты не принимают активного участия в разработке международных стандартов, но имеют полный доступ к интересующей их информации. Наконец, есть еще и члены-подписчики – они представлены организациями развивающихся стран. В настоящее время Россия имеет статус члена-корреспондента ISO.

Выполнение технической работы в ISO возложено на 2700 технических комитетов, подкомитетов и рабочих групп, в состав которых входят представители правительственных, промышленных, научно-исследовательских и юридических кругов (на сегодняшний день – около 500 организаций). Каждая организация – член ISO – имеет право включить своего представителя в любой комитет, если она особо заинтересована в деятельности этого комитета.

Работу комитетов курируют крупнейшие организации – члены ISO: AFNOR, ANSI, BSI, CSBTS, DIN, SIS и др. Принятие стандарта возможно только после достижения консенсуса между квалифицированным большинством членов комитета.

Центральный секретариат ISO, штаб-квартира которого находится в Женеве, публикует исходные версии стандартов, подготовленные комитетами, и рассылает их членам ISO для ознакомления и голосования. Регулярно издаваемый бюллетень «ISO Momento» содержит подробную информацию о структуре и деятельности всех технических комитетов. Круг интересов и сфера деятельности ISO не ограничивается какой-либо конкретной технологической областью. Организацию интересует все и вся, за исключением электротехники и электроники (эти направления контролирует International Electrotechnical Commission – IEC). Работы в области информационных технологий ведутся ISO и IEC (Рабочий комитет JTC 1) совместно, поэтому соответствующие стандарты имеют маркировку ISO/IEC.

Новые стандарты рождаются в соответствии с тремя принципами.

Во-первых, они являются результатом *консенсуса всех заинтересованных сторон* – производителей, поставщиков, потребителей, профессиональных разработчиков, правительственных и исследовательских организаций.
Во-вторых, стандарты имеют действительно *мировое распространение* и удовлетворяют *как производителей, так и потребителей*.
В-третьих, появление новых стандартов диктуется исключительно *научными достижениями, появлением новых технологий, а также требованиями свободного рынка*, а не чьей-то злой или доброй волей. Если рынок созрел для нового стандарта, то такой стандарт появляется.

Процесс создания нового стандарта включает три этапа. Обычно инициатива его разработки исходит от производителей, которые доводят базовые предложения стандарта до своего представителя в ISO. Если эта организация признает целесообразность появления нового стандарта, то соответствующая рабочая группа определяет техническую область, на которую предполагаемый стандарт будет распространяться. На втором этапе идет выработка технических спецификаций, в ходе которой представители различных стран стремятся к достижению консенсуса. На заключительном этапе первая версия стандарта утверждается (за стандарт должно проголосовать 75% кворума) и публикуется. *С этого момента стандарт становится официальным International Standard of ISO!*

По мере совершенствования технологий, появления новых материалов, методов обработки, повышения требований к качеству и надежности изделий возникает необходимость в пересмотре стандартов. В ISO существует правило: все стандарты должны пересматриваться не реже одного раза в пять лет. Сегодня «перу» ISO принадлежит около 9300 различных стандартов, описание которых занимает 171000 страниц текста на английском языке.

Специальная группа планирования работ подкомитета № 7 ISO в 1995 г. разработала перспективную Программу развития стандартизации (ПРС) в области программной инженерии (Software Engineering Programme), где дается обзор общих проблем и требований к программной инженерии, а также пути их решения в

пределах системы международных стандартов. Определено, что стандарты должны:

- ориентироваться на процессы, продукты, ресурсы и терминологию в области разработки программных средств (ПС);
- способствовать распространению продуктов с предсказуемым качеством;
- способствовать повышению производительности и качества труда программистов;
- содействовать воспроизводимости процессов для ПС;
- поддерживать оценку соответствия ПС заданным требованиям объективными методами;
- инициировать или мотивировать создание необходимой инструментальной поддержки;
- производить экономический эффект при применении;
- защищать общественную безопасность, здоровье и благосостояние людей;
- основываться на реалиях, а не на непроверенных теориях.

Перечень стандартов ISO для построения системы качества и управления качеством

1. ISO 9000-1 (1994 г.). Управление качеством и гарантии качества. Часть 1. Руководство по выбору и использованию.
2. ISO 9000-2 (1993 г.). Управление качеством и гарантии качества. Часть 2. Общее руководство по применению стандартов ISO 9001, ISO 9002 и ISO 9003.
3. ISO 9000-3 (1991 г.). Управление качеством и гарантии качества. Часть 3. Руководство по применению стандарта ISO 9001 при разработке, установке и сопровождении ПО.
4. ISO 9000-4 (1993 г.). Управление качеством и гарантии качества. Часть 4. Руководство по управлению надежностью программ. ISO 9001 (1994 г.).
5. Системная модель качества для процессов проектирования, разработки, производства, установки и обслуживания. ISO 9002 (1994 г.).
6. Системная модель качества для процессов проверки качества проектирования, установки и обслуживания. ISO 9003 (1993 г.).

7. Системная модель качества для процессов проверки качества при окончательном тестировании.
8. ISO 10011-1 (1990 г.). Руководство по аудиту качества систем. Часть 1. Аудит.
9. ISO 10011-2 (1991 г.). Руководство по аудиту качества систем. Часть 2. Квалификационные требования, предъявляемые к аудиторам качества.
10. ISO 10011-3 (1991 г.). Руководство по аудиту качества систем. Часть 3. Менеджмент программ аудита.
11. ISO 10012-1 (1992 г.). Требования по качеству, предъявляемые к измерительной аппаратуре. Часть 1.
12. ISO 10013. Руководство по созданию качественной документации.
13. ISO/TR 13425. Руководство по выбору статистических методов, используемых при разработке стандартов и спецификаций.
14. ISO 8402 (1994 г.). Управление качеством и гарантии качества. Словарь терминов.
15. ISO 9000:2000. Системы менеджмента качества, в 4-х частях.

Для прохождения процедуры сертификации в ISO организации должны представить комплект документации, оформленной стандартным способом (Quality System Documentation). Задача эта ДОСТАТОЧНО сложная и отнимает много времени – чтобы упростить ее, можно воспользоваться программой ISOplus, выпускаемой Software Productivity Center. Это комплект более чем из 100 файлов, представленных в формате Word и выполняющих роль шаблонов документов, процедур, анкет и руководств, которые пригодятся в ходе подготовки к сертификации.

Стандарт ISO/IEC 9126:1991 «Оценивание программного продукта. Характеристики качества и руководящие указания по их применению»

В совместном стандарте ISO и Международной комиссии по электротехнике (IEC) ISO/IEC 9126:1991 определены шесть групповых характеристик верхнего уровня: *функциональность* (Functionality), *надежность* (Reliability), *удобство использования* (Usability), *эффективность* (Efficiency), *сопровождаемость* (Maintainability), *переносимость* (Portability) и дан

предварительный перечень групповых характеристик второго уровня иерархии (подхарактеристик) [15.7, 15.8]. Стандарт, таким образом, дал реальную возможность для развития работ по установлению и стандартизации полной номенклатуры показателей качества вплоть до единичных измеряемых показателей (метрик).

В более поздней версии 1993 года ISO/IEC 9126:1993 выделены несколько видоизмененные характеристики (показатели) качества с позиций пользователя, разработчика и управляющего проектом. Документом рекомендуется шесть основных характеристик верхнего уровня: *функциональная пригодность, надежность, применимость, эффективность, сопровождаемость и переносимость*, детализированные 21 показателем второго уровня. Ниже приводятся смысловое содержание некоторых важных показателей первого и второго уровней [15.9]:

- *Функциональная возможность* – способность программного средства обеспечивать решение задач, удовлетворяющих сформулированные потребности заказчиков и пользователей при применении комплекса программ в заданных условиях.
- *Функциональная пригодность* – набор и описания субхарактеристики и ее атрибутов, определяющие назначение, номенклатуру, основные, необходимые и достаточные функции программного средства, соответствующие техническому заданию и спецификациям требований заказчика или потенциального пользователя.
- *Правильность (корректность)* – способность программного средства обеспечивать правильные или приемлемые для пользователя результаты и внешние эффекты.
- *Способность к взаимодействию* – свойство программных средств и их компонентов взаимодействовать с одной или большим числом компонентов внутренней и внешней среды.
- *Защищенность* – способность компонентов программного средства защищать программы и информацию от любых негативных воздействий.
- *Надежность* – обеспечение комплексом программ достаточно низкой вероятности отказа в процессе функционирования программного средства в реальном времени.
- *Эффективность* – свойства программного средства,

обеспечивающие требуемую производительность решения функциональных задач, с учетом количества используемых вычислительных ресурсов в установленных условиях.

- *Практичность (применимость)* – свойства программного средства, обусловливающие сложность его понимания, изучения и использования, а также привлекательность для квалифицированных пользователей при применении в указанных условиях.
- *Сопровождаемость* – приспособленность программного средства к модификации и изменению конфигурации и функций.
- *Мобильность (переносимость)* – подготовленность программного средства к переносу из одной аппаратно-операционной среды в другую.

В настоящее время завершается разработка и формализован последний проект состоящего из четырех частей стандарта ISO 9126-(1-4) для замены редакций 1991 и 1993 годов. Проект состоит из следующих частей под общим заголовком «Информационная технология – характеристики и метрики качества программного обеспечения»:

- Часть 1. Характеристики и субхарактеристики качества
- Часть 2. Внешние метрики качества
- Часть 3. Внутренние метрики качества
- Часть 4. Метрики качества в использовании.

Первая часть стандарта ISO 9126-1 – распределяет атрибуты качества программных средств по шести характеристикам, используемым в остальных частях стандарта. Исходя из принципиальных возможностей их измерения, все характеристики могут быть объединены в три группы, к которым применимы разные категории метрик:

- *категорийные (описательные, номинальные) метрики* предназначены для «измерения» функциональных возможностей программных средств;
- *количественные метрики* применимы для измерения надежности и эффективности сложных комплексов программ;
- *качественные метрики* в наибольшей степени соответствуют практичности, сопровождаемости и мобильности программных

средств.

В этой части стандарта ISO 9126-1 даются также определения с уточнениями из остальных его частей для каждой характеристики программного средства, а также для субхарактеристик качества.

Вторая и третья части стандарта ISO 9126-2 и ISO 9126-3 посвящены формализации соответственно внешних и внутренних метрик характеристик качества сложных программных средств. Все таблицы содержат унифицированную рубрикацию, где отражены имя и назначение метрики; метод ее применения; способ измерения, тип шкалы метрики; тип измеряемой величины; исходные данные для измерения и сравнения; а также этапы жизненного цикла программного средства (по ISO 12207), к которым применима метрика.

Четвертая часть стандарта ISO 9126-4 предназначена для покупателей, поставщиков, разработчиков, сопровождающих, пользователей и менеджеров качества программных средств. В ней обосновываются и комментируются выделенные показатели сферы использования (контекста) программных средств и группы выбранных метрик для пользователей.

13.5. Методика выбора показателей качества

В начале этого параграфа уместно привести две цитаты, которые исчерпывающе «закрывают» вопрос о необходимости формирования систем количественных и качественных показателей для адекватного измерения процессов разработки и качества продукта: «Когда вы можете измерить, то о чем вы говорите, и выразить это в числах – вы знаете кое-что об этом; но если вы не можете измерить это и выразить в числах – ваше знание скудно и неудовлетворительно» (Лорд Кельвин) и «Мы не можем управлять тем, что мы не можем измерить» (Том де Марко).

Действительно, реально и эффективно управлять процессом разработки можно лишь в том случае, если в каждый момент времени мы можем судить о том, насколько правильно он выполняется. А это возможно только в том случае, если мы имеем систему показателей (метрик) для измерения параметров процесса и

можем соотносить их величину и колебания с запланированными или модельными величинами [10].

Проблема формирования адекватных систем показателей действительно является сложной проблемой. Конечно, имеется достаточно много простых и очевидных внешних метрик, единицами измерения которых являются время, деньги, процентные соотношения. Однако, как только речь заходит об измерении внутренних показателей большого проекта или комплексного процесса разработки программного продукта, мы вступаем в область «ноу-хау», так как и проекты и процессы, с которыми имеют дело в различных компаниях, часто являются уникальными, и компании неохотно делятся своими секретами. Именно поэтому достаточно известно о стандартных метриках (ISO 9126), и так мало публикаций о корпоративных достижениях в области метрологии процессов разработки программного обеспечения.

Исходными данными и высшим приоритетом при выборе показателей качества в большинстве случаев являются *назначение, функциональная пригодность и безопасность* соответствующего программного средства. Достаточно полное и корректное описание этих свойств должно служить базой для определения значений большинства остальных характеристик и атрибутов качества. Принципиальные и технические возможности и точность измерения значений характеристик качества всегда ограничены в соответствии с их содержанием. Это определяет рациональные диапазоны значений каждого атрибута, которые могут быть выбраны на основе здравого смысла, а также путем анализа прецедентов в спецификациях системных и функциональных требований реальных проектов.

Критерии качества программного продукта отражают следующие его характе-ристики: *обоснованность, надёжность, модульность, тестируемость, переносимость, гибкость, модифицируемость, документированность, ясность, точность, эффективность, экономичность, легкость сопровождения* и т. д. (Рис. 13.5.1). Такой критерий должен:

- численно характеризовать набор основных целевых функций программного средства;
- обеспечивать возможность определения общих и конкретных

затрат, необходимых для достижения требуемого качества;

- оценивать степени влияния на показатель качества различных внешних факторов;
- быть по возможности простым, хорошо измеримым и иметь малую дисперсию на широком диапазоне измерений.

Рис. 13.5.1. Критерии качества (Мак-Кол, Ричардс, Уолтерс, 1977 г.)

Указанные выше составляющие качества не появляются сами собой – они складываются из многих факторов. На Рис. 13.5.2 показаны слагаемые качества программного продукта – и каждое из этих слагаемых должно быть тщательно измерено.

Определим понятие метрика. Под метрикой мы понимаем *текстовые, символьные, формульные, табличные формализованные оценки, факторы и критерии, а также правила их применения, образующие базовую систему для измерений различных параметров как самих компьютерных программ,*

процессов, сервисов, инструментов и средств разработки, ресурсов, так и их качества. Эти измерения могут проводиться на уровне общих критериев качества или на уровне отдельных характеристик стандартного процесса, текущего проекта, разрабатываемого продукта.

Рис. 13.5.2. Слагаемые качества программного продукта.

В настоящее время в мировой практике используется несколько сотен метрик компьютерных программ [11]. Существующие качественные и количественные оценки можно сгруппировать по шести основным направлениям:

1. оценки топологической и информационной сложности программ;
2. оценки уровня языковых средств и их применения;
3. оценки трудности восприятия и понимания программных текстов, ориентированные на психологические факторы, существенные для сопровождения и модификации программ;
4. оценки надежности программных систем, позволяющие прогнозировать нештатные и «отказовые» ситуации;
5. оценки производительности ПО и повышения его эффективности путем выявления и коррекции ошибок концептуального и системного проектирования;

6. оценки производительности труда программистов для прогнозирования сроков разработки программ и планирования работ по созданию программных комплексов.

Структура метрики может быть достаточно сложной – в общем случае элементами метрики являются следующие базовые составляющие:

- используемая модель технологии производственного процесса (Production Technology Model – PTM);
- используемая модель внешней среды, в которой протекает деятельность рассматриваемой процессной или проектной единицы (Environment Model – EM);
- цели, преследуемые в ходе производственной деятельности (Activity Goals – AG);
- используемая модель функционирования процессно-проектной единицы в рассматриваемой среде – модель реализации имеющихся технологических возможностей производства (Performance Model – PM);
- выбранный метод измерения эффективности (Measurement Model – MM);
- выбранная шкала измерения эффективности (Units of Measure – UM).

Процессы выбора и установления метрик и шкал для описания характеристик качества программных средств можно разделить на два общих этапа:

- выбор и обоснование набора исходных данных, отражающих общие особенности и этапы жизненного цикла проекта программного средства и его потребителей, каждый из которых влияет на определенные характеристики качества комплекса программ;
- выбор, установление и утверждение конкретных метрик и шкал измерения характеристик и атрибутов качества проекта для их последующей оценки и сопоставления с требованиями спецификаций в процессе квалификационных испытаний или сертификации на определенных этапах жизненного цикла программного средства.

На первом этапе за основу следует брать всю базовую номенклатуру характеристик, субхарактеристик и атрибутов,

стандартизированных в ISO 9126. Их описания желательно предварительно упорядочить по приоритетам с учетом назначения и сферы применения конкретного проекта программного средства. Далее необходимо выделить и ранжировать по приоритетам потребителей, которым необходимы определенные показатели качества проекта программного средства с учетом их специализации и профессиональных интересов. Подготовка исходных данных завершается выделением номенклатуры базовых, приоритетных показателей качества, определяющих функциональную пригодность программного средства для определенных потребителей и общую систему количественных показателей, которые будет использовать разработчик.

На втором этапе, после фиксирования исходных данных, которое должен выполнить потребитель оценок качества, процесс выбора метрик начинается с ранжирования характеристик и субхарактеристик для конкретного проекта, процесса, продукта и их потребителя. Далее этими специалистами для каждого из отобранных показателей должна быть установлена и согласована метрика и шкала оценок субхарактеристик и их атрибутов для проекта и потребителя результатов анализа. Определяются измеряемые зоны процессов, устанавливаются перекрытия зон, оцениваются примерные величины корреляций измеряемых величин.

На основании этого можно сформировать конкретный алгоритм формирования метрики:

1. Определение множества характеристик, которые, являясь важными для программного обеспечения, допускают несложное измерение и не перекрываются.

2. Выделение кандидатов в метрики, которые измеряют степень удовлетворения указанным характеристикам.

3. Исследование характеристик и связанных метрик, для определения корреляции, значимости, степени автоматизируемости измерений.

4. Анализ и нахождение корреляции между метриками, степени перекрытия, зависимости и недостатки.

5. Рафинирование множества метрик в целом – во множество метрик, которые в совокупности адекватно отражают качество программного обеспечения.

6. Корректировка каждой метрики в итоговом множестве в контексте зафиксированных множеств характеристик и метрик.

Для показателей, представляемых качественными признаками, желательно определить и зафиксировать в спецификациях описания условий, при которых следует считать, что данная характеристика реализуется в программном средстве. Выбранные значения характеристик качества и их атрибутов должны быть предварительно проверены разработчиками на их реализуемость с учетом доступных ресурсов конкретного проекта и при необходимости откорректированы.

13.6. Оценка качества программного продукта по ISO 14598

Методологии и стандартизации оценки характеристик качества готовых программных средств и их компонентов (программного продукта) на различных этапах жизненного цикла посвящен международный стандарт ISO 14598, состоящий из шести частей. Стандарт рекомендует следующую общую схему оценки характеристик качества программ:

- установка исходных требований для оценки – определение целей испытаний, идентификация типа метрик программного средства, выделение адекватных показателей и требуемых значений атрибутов качества;
- селекция метрик качества, установление рейтингов и уровней приоритета метрик субхарактеристик и атрибутов, выделение критериев для проведения экспертиз и измерений;
- планирование и проектирование процессов оценки характеристик и атрибутов качества в жизненном цикле программного средства;
- выполнение измерений для оценки, сравнение результатов с критериями и требованиями, обобщение и оценка результатов.

Для каждой характеристики качества рекомендуется формировать меры и шкалу измерений с выделением требуемых,

допустимых и неудовлетворительных значений. Реализация процессов оценки должна коррелировать с этапами жизненного цикла конкретного проекта программного средства в соответствии с адаптированной версией стандарта ISO 12207.

Стандарт ISO 14598 рассматривает и оценивает следующие характеристики качества ПС:

Оценка функциональной пригодности – наиболее неопределенная и объективно трудно оцениваемая субхарактеристика программного средства. Области применения, номенклатура и функции комплексов программ охватывают столь разнообразные сферы деятельности человека, что невозможно выделить и унифицировать небольшое число атрибутов для оценки и сравнения этой субхарактеристики в различных комплексах программ. Тем не менее, стандарт выделяет некоторые обобщенные субхарактеристики.

Оценка корректности программных средств состоит в формальном определении степени соответствия комплекса реализованных программ исходным требованиям контракта, технического задания и спецификаций на программное средство и его компоненты. Путем верификации должно быть определено соответствие исходным требованиям всей совокупности к компонентов комплекса программ, вплоть до модулей и текстов программ и описаний данных.

Оценка способности к взаимодействию состоит в определении качества совместной работы компонентов программных средств и баз данных с другими прикладными системами и компонентами на различных вычислительных платформах, а также взаимодействия с пользователями в стиле, удобном для перехода от одной вычислительной системы к другой с подобными функциями.

Оценка защищенности программных средств включает определение полноты использования доступных методов и средств защиты программного средства от потенциальных угроз и достигнутой при этом безопасности функционирования информационной системы. Наиболее широко и детально методологические и системные задачи оценки комплексной защиты информационных систем изложены в трех частях стандарта ISO

15408:1999-(1-3) «Методы и средства обеспечения безопасности. Критерии оценки безопасности информационных технологий».

Оценка надежности – измерение количественных метрик атрибутов субхарактеристик в использовании: завершенности, устойчивости к дефектам, восстанавливаемости и доступности/готовности.

Оценка потребности в ресурсах памяти и производительности компьютера в процессе решения задач значительно изменяется в зависимости от состава и объема исходных данных. Для корректного определения предельной пропускной способности информационной системы с данным программным средством нужно измерить экстремальные и средние значения длительностей исполнения функциональных групп программ и маршруты, на которых они достигаются. Если предварительно в процессе проектирования производительность компьютера не оценивалась, то, скорее всего, понадобится большая доработка или даже замена компьютера на более быстродействующий.

Оценка практичности программных средств проводится экспертами и включает определение понятности, простоты использования, изучаемости и привлекательности программного средства. В основном это качественная (и субъективная) оценка в баллах, однако некоторые атрибуты можно оценить количественно по трудоемкости и длительности выполнения операций при использовании программного средства, а также по объему документации, необходимой для их изучения.

Оценка сопровождаемости производится полнотой и достоверностью документации о состояниях программного средства и его компонентов, всех предполагаемых и выполненных изменениях, позволяющей установить текущее состояние версий программ в любой момент времени и историю их развития. Она должна определять стратегию, стандарты, процедуры, распределение ресурсов и планы создания, изменения и применения документов на программы и данные.

Оценка мобильности – качественное определение экспертами адаптируемости, простоты установки, совместимости и замещаемости программ, выражаемое в баллах. Количественно эту характеристику программного средства и совокупность ее

атрибутов можно (и целесообразно) оценить в экономических показателях: стоимости, трудоемкости и длительности реализации процедур переноса на иные платформы определенной совокупности программ и данных.

13.7. Процессный подход и формирование системы качества при разработке ПО

Выбор характеристик и оценка качества программных средств – лишь одна из задач в области обеспечения качества продукции, выпускаемой компаниями-разработчиками ПО. Комплексное решение задач обеспечения качества программных средств предполагает разработку и внедрение той или иной системы управления качеством.

В мировой практике наибольшее распространение получила система, основанная на международных стандартах серии ISO 9000, включающей десяток с лишним документов, в том числе стандарт, регламентирующий обеспечение качества ПО (ISO 9000-3) [12, 13]. Эти стандарты, а также специализированные стандарты ISO 12207, CMM, SPICE должны служить руководством для ведущих специалистов компаний, разрабатывающих ПО на заказ.

Стандарт ISO/IEC 12207:1995 «Информационные технологии. Процессы жизненного цикла программного обеспечения».

Первая редакция ISO12207 подготовлена в 1995 году объединенным техническим комитетом ISO/IEC JTC1 «Информационные технологии, подкомитет SC7, проектирование программного обеспечения». По определению, ISO12207 – базовый стандарт процессов жизненного цикла программного обеспечения (ЖЦ ПО), ориентированный на различные (любые!) виды ПО и типы проектов разработки информационных систем, куда ПО входит как часть. Стандарт определяет стратегию и общий порядок в создании и эксплуатации ПО, он охватывает ЖЦ ПО от концептуализации идей до завершения ЖЦ [14].

Важное замечание: процессы, используемые во время ЖЦ разработки ПО, должны быть совместимы с процессами, используемыми во время ЖЦ создания информационной системы. (Отсюда понятна целесообразность совместного использования

стандартов на ИС и на ПО. В связи с этим в стандарте дано следующее определение: система – это объединение одного или более процессов, программного обеспечения, аппаратных средств, телекоммуникационного и другого оборудования и персонала для обеспечения возможности удовлетворения определенных потребностей или целей).

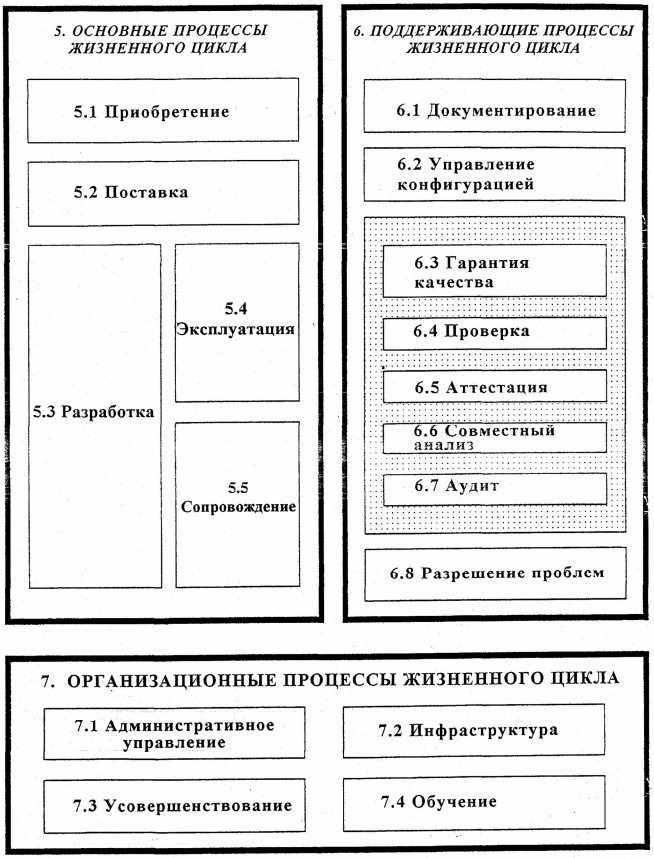


Рис. 13.7.1. Процессная область стандарта ISO 12207.

Стандарт ISO12207 равносильно ориентирован на организацию действий каждой из двух участвующих сторон: поставщик (разработчик) и покупатель (пользователь). Стандарт может быть в

равной степени применен и в том случае, когда обе стороны – из одной организации.

Общая структура стандарта. Процессы ЖЦ. По сравнению с известными стандартами ISO 12207 состоит из гораздо более крупных обобщенных процессов: «приобретение», «поставка», «разработка» и т. п. (рис. 15.5). Каждый процесс разделен на набор действий, каждое действие – на набор задач (рис. 15.6). Очень важное отличие ISO 12207 от стандартов серии ISO 9000: каждый процесс, действие или задача инициируется и выполняется другим процессом по мере необходимости, причем нет заранее определенных последовательностей (естественно, при сохранении логики связей по исходным сведениям задач и т. п.) – всё определяется конкретным проектом разработки.

Структура процессов. В стандарте выделено 5 основных процессов жизненного цикла разработки ПО:

1. *Процесс приобретения*. Определяет действия предприятия-покупателя, которое приобретает АС, программный продукт или сервис ПО.

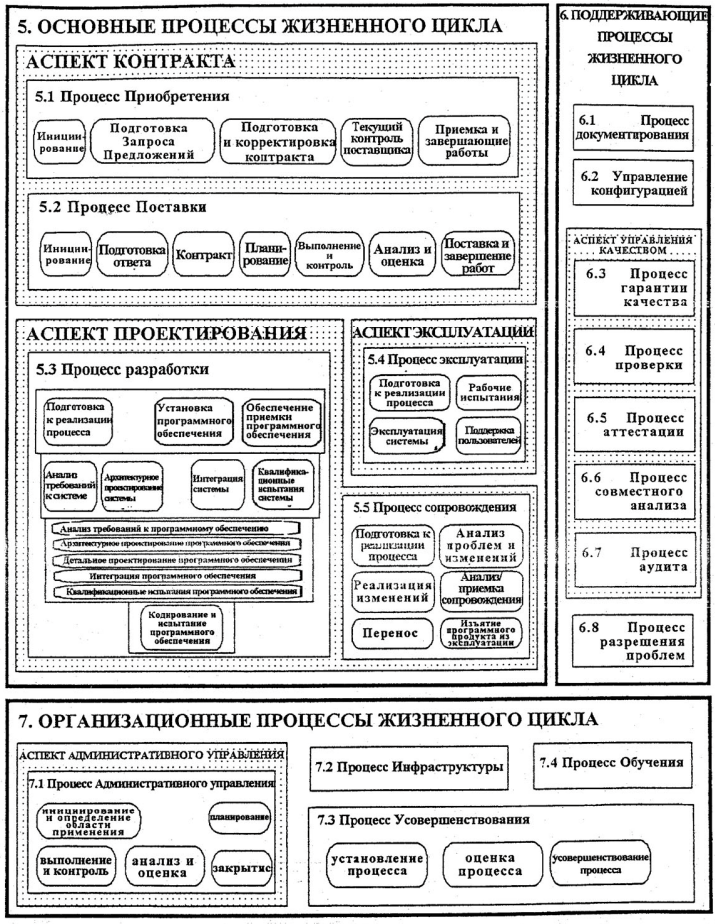
2. *Процесс поставки*. Определяет действия предприятия-поставщика, которое снабжает покупателя системой, программным продуктом или сервисом ПО.

3. *Процесс разработки*. Определяет действия предприятия-разработчика, которое разрабатывает принцип построения программного изделия и программный продукт.

4. *Процесс функционирования (эксплуатации)*. Определяет действия предприятия-оператора, которое обеспечивает обслуживание системы (а не только ПО) в процессе ее функционирования в интересах пользователей. В отличие от действий, которые определяются разработчиком в инструкциях по эксплуатации (эта деятельность разработчика предусмотрена во всех трех рассматриваемых стандартах), определяются действия оператора по консультированию пользователей, получению обратной связи и др., которые он планирует сам и берет на себя соответствующе обязанности.

5. *Процесс сопровождения*. Определяет действия персонала сопровождения, который обеспечивает сопровождение программного продукта, что представляет собой управление

модификациями программного продукта, поддержку его текущего состояния и функциональной пригодности, включает в себя инсталляцию и удаление программного изделия на вычислительной системе.



Порядок расположения работ не означает временного порядка их выполнения

Название работ в Процессе Разработки не являются названиями этапов разработки

Рис. 15.7.2. Аспекты и процессы разработки.

В стандарте обозначены восемь вспомогательных (поддерживающих) процессов, которые сопровождают реализацию другого процесса, будучи неотъемлемой частью всего ЖЦ программного изделия, и обеспечивают должное качество проекта ПО. Вспомогательные процессы – это процессы решения проблем, документирования, управления конфигурацией, гарантирования качества, последний из которых использует результаты остальных процессов группы обеспечения качества, в которую входят: процесс верификации, процесс аттестации, процесс совместной оценки, процесс аудита.

Число организационных процессов – четыре: это процессы управления, создания инфраструктуры, усовершенствования, обучения. К ним примыкает особый *процесс адаптации*, который определяет основные действия, необходимые для адаптации стандарта к условиям конкретного проекта.

Под процессом усовершенствования здесь понимается не усовершенствование ИС или ПО, а улучшение самих процессов приобретения, разработки, гарантирования качества и т. п., реально осуществляемых в организации.

Особенности стандарта ISO 12207. «Динамический» характер стандарта определяется способом определения последовательности выполнения процессов и задач, при котором один процесс при необходимости вызывает другой или его часть. Примеры:

- выполнение Процесса приобретения в части анализа и фиксации требований к системе или ПО может вызывать исполнение соответствующих задач Процесса разработки;
- в Процессе поставки поставщик должен управлять субподрядчиками согласно Процессу приобретения и выполнять верификацию и аттестацию по соответствующим процессам;
- сопровождение может требовать развития системы и ПО, что выполняется по Процессу разработки.
- Такой характер взаимосвязи процессов позволяет реализовывать любую модель ЖЦ.

Таким образом, для всех этапов жизненного цикла программных продуктов разработчик должен самостоятельно разрабатывать комплексы показателей качества, которые в совокупности образуют систему показателей (метрик). Он также

самостоятельно должен выявлять факторы, влияющие на качество. Только такой структурированный индивидуальный подход к выбору и обоснованию показателей и факторов обеспечивает эффективный контроль и управление качеством.

Версия международного стандарта ISO 9000:2000.

Стандарты ISO являются наиболее известными и распространенными в мире. Они универсальны, их можно применять в качестве моделей независимо от отрасли, в которой функционирует компания. Такой подход, очевидно, имеет свои преимущества и недостатки.

Основным преимуществом международность стандартов ISO серии 9000 является их легитимность, известность, распространенность, признание на мировом уровне, большое количество экспертов и аудиторов и невысокая стоимость услуг по сертификации. Универсальность же моделей ISO серии 9000 содержит в себе определенные недостатки: они являются достаточно высокоуровневыми, задают абстрактные модели и не содержат конкретных методологических разработок. Стандарты носят рекомендательный характер, выполнение их требований является сугубо добровольным делом разработчика.

По ISO «качество – это совокупность свойств и характеристик продукта, процесса или услуги, которые обеспечивают способность удовлетворять заявленным или подразумеваемым потребностям». Современные способы обеспечения качества базируются на принципах Всеобщего Руководства Качеством (Total Quality Management – TQM). TQM – это управление ресурсами и применение количественных методов анализа для улучшения:

- разработок, материалов и услуг, поставляемых в организацию;
- всех процессов внутри организации;
- степени удовлетворенности настоящих и будущих потребностей клиентов.

Задавая модели системы качества, стандарты семейства ISO 9000 *лишь формулируют требования*, которые должны быть реализованы, но не говорится о том, *как это можно сделать*. Это связано с тем, что для того, чтобы рекомендовать абстрактному предприятию способы реализации и зафиксировать необходимые

рекомендации по выполнению требований, необходимо конкретизировать:

- сферу деятельности организации;
- специфику ее процессов;
- специфику культуры предприятия;
- структуру управления, существующую в компании (матричная, иерархическая проектная) и другие особенности, свойственные компании.

Поэтому для построения полноценной системы качества по ISO необходимо помимо основной модели ISO 9001:2000, необходимо использовать вспомогательные отраслевые и рекомендательные стандарты. Для организации, занимающейся разработкой программного обеспечения, таким стандартами являются: ISO 9000-3:91, ISO 10007:95, ISO 10013:95, ISO 12207:95.

Рис.13.7.3. Процессный подход – основа реализации на базе требований ISO 9000:2000.

Семейство стандартов ISO 9000 версии 2000 года разработано с тем, чтобы преодолеть недостатки ISO 9000 версии 1994 года и

помочь организациям всех специализаций, типов и размеров внедрить и использовать эффективные системы менеджмента качества на базе внедрения и повседневного использования процессного подхода (Рис. 13.7.3).

Семейство стандартов включает следующие документы:

- ISO 9000:2000, являющийся введением в системы менеджмента качества и содержащий соответствующий словарь;
- ISO 9001:2000, устанавливающий детальные требования для систем менеджмента качества;
- ISO 9004:2000, обеспечивающий руководство по развитию системы менеджмента качества;
- ISO 10011:2000, предназначенный для управления и проведения внутреннего и внешнего аудитов системы менеджмента качества.

Эти стандарты различают требования к системам менеджмента качества и требования к программным продуктам, как, впрочем, и к любой другой продукции. Требования к системам менеджмента качества подробно определены в стандарте ISO 9001:2000.

Подход к системам менеджмента качества является общим и применяется к организациям в любой отрасли экономики, поэтому данный стандарт не устанавливает каких-либо конкретных требований к программным продуктам. Требования к ним могут определяться заказчиками или третьими лицами и содержаться в технических спецификациях, стандартах на продукт, стандартах на процесс, контрактных соглашениях и нормативных документах.

В основу построения системы качества в соответствии с моделью ISO 9000:2000 закладываются следующие 8 основных принципов:

- концентрация на потребностях заказчика;
- активная лидирующая роль руководства;
- вовлечение исполнителей в процессы совершенствования;
- реализация процессного подхода;
- системный подход к управлению;
- обеспечение непрерывных улучшений;
- принятие решений на основе фактов;
- взаимовыгодные отношения с поставщиками.

При этом методически в полном соответствии с дисциплиной построения сложных систем в стандарте ISO 9000:2000 предусматривается с одной стороны построение организационной системы «сверху - вниз»: от целей предприятия и его политики – к организационной структуре и формированию бизнес процессов, и с другой – итеративное развитие организационной системы через механизмы измерения и улучшения.

Внедрение системы менеджмента качества организацией – разработчиком программных продуктов по ISO 9000 версии 2000 года состоит из нескольких этапов. В их числе можно выделить выделить:

- измерение характеристик продуктов для определения эффективности каждого процесса, направленного на достижение целей качества;
- применение результатов измерений для определения текущей эффективности процессов создания и внедрения продуктов;
- определение способов предотвращения дефектов, снижения изменяемости продукции и минимизации доработок;
- поиск возможностей по снижению рисков и улучшению эффективности и производительности технологических и иных процессов;
- выявление и расстановка в порядке важности тех улучшений, которые могут давать оптимальные результаты с приемлемыми рисками;
- планирование стратегии, процессов и ресурсов для получения идентифицированных улучшений продукции;
- контроль результатов улучшений;
- сравнение полученных результатов с ожидаемыми или запланированными;
- определение подходящих корректирующих действий.

Реализация этих этапов возможна только при наличии в организации системы критериев, показателей и факторов качества, а также методов их измерения и оценки.

Система менеджмента качества является частью системы управления, которая ориентирована на достижение результатов, основанных на целях качества, удовлетворении нужд и ожиданий заказчиков. Цели качества дополняют другие цели организации.

Различные части системы управления организации-разработчика могут быть объединены вместе с системой менеджмента качества в единую, унифицированную систему управления с общими элементами. Это способствует планированию, распределению ресурсов, установлению взаимодополняющих целей и оценке эффективности.

Стандарт TickIT

Достаточно широкую известность в среде разработчиков ПО получил британский стандарт TickIT (tick – оболочка, чехол) [15]. Как уже было сказано выше, для построения системы качества такой компании абсолютно недостаточно использовать только модель ISO 9001, а необходимо воспользоваться группой специализированных и рекомендательных стандартов. Попытка представить консолидированную модель и соответствующие дать рекомендации, составленные из необходимых стандартов для разработчиков программного обеспечения, реализована в стандарте TickIT.

В отличие от модели ISO 9001, которая регламентирует «*что необходимо сделать?*» разработчики данного стандарта попытались ответить на вопрос *«что необходимо сделать для становления системы качества в организации, занимающейся разработкой программного обеспечения?»*.

TickIT – это ещё и схема сертификации систем качества для программного обеспечения, предложенная группой ведущих фирм и некоммерческих организаций Великобритании, работающих в области информатики. TickIT базируется на стандарте ISO 9001:1994. Таким образом, предметом TickIT является менеджмент предприятий, разрабатывающих программное обеспечение.

Согласно схеме TickIT могут быть сертифицированы системы качества предприятий, занимающихся следующими видами деятельности:

- разработка программного продукта или услуги как для внешнего заказчика, так для внутреннего использования, включая встроенное (Embedded) программное обеспечение;
- системная интеграция, поддержка, администрирование программных комплексов и систем;

- копирование, архивирование, хранение данных и программного обеспечения.

Помимо своей основы – стандарта ISO 9001, TickIT содержит следующие дополнительные компоненты:

- руководство по TickIT, базирующееся на указаниях ISO 9000-3 (руководство по системам качества для программного обеспечения);
- схему регистрации аудиторов через специальный Комитет по TickIT IRCA (Международный Регистр Сертифицированных Аудиторов);
- систему проверок аудиторов Британским Компьютерным Обществом (BCS) и Институтом по Обеспечению Качества (IQA);
- систему аккредитации сертификационных обществ UKAS (Великобритания), SWEDAC (Швеция);
- программы, направленные на расширение признания схемы;
- трехлетний цикл пересмотра схемы;
- систему специальных премий за достижения.

Стандарт TickIT кроме требований модели ISO 9001 включает набор требований и практик из стандартов ISO 12207 «Жизненный цикл ПО» и ISO 9000-3 «Руководство для внедрения ISO 9001:94 для разработки, установки и поддержки ПО». Преимуществом данной модели является в первую очередь то, что данный стандарт представляет собой *достаточную модель*, по которой можно не просто проверять, а самое главное разрабатывать систему качества. Следующее важное преимущество – разрабатывая систему качества по модели TickIT, мы также строим «ISO 9000-совместимую» систему качества компании.

Стандарт «Использование ISO 9001:2000 для построения систем менеджмента качества программных продуктов, сертификации и непрерывного улучшения» (The TickIT Guide) разработан профессионалами отрасли из Европы и США, приглашёнными для работы над стандартом в составе специального комитета BRD/3/1 Британским Институтом Стандартизации (BSI).

К недостаткам модели TickIT можно отнести недостаточное распространение данного стандарта и, как следствие, недостаточное его признание в мире. В самом деле – если рассмотреть стандарты,

которые были взяты в основу TickIT, то невольно задаешься вопросом, почему разработчики TickIT не включили в свою модель другие не менее важные и нужные для организаций разработчиков программного обеспечения стандарты. Такие как ISO 9004-1, ISO 10007 «Управление конфигурацией» и другие?

Здесь нужно учитывать следующее важное обстоятельство – модель TickIT создавалась как отраслевой стандарт для внедрения модели ISO 9000 в организациях разработчиках ПО, и создатели данного стандарта дали своё, далеко не бесспорное представление о необходимости ограничится лишь тремя из всего многообразия справочных и рекомендательных стандартов для построение эффективной системы качества.

Отметим всё же, что для первого представления о системе качества для разработчиков ПО гораздо проще прочесть изучить TickIT, а не разбираться во всей сложной структуре иерархии рекомендательных стандартов ISO в области обеспечения качества. Гибкая инфраструктура TickIT позволяет схеме следовать изменениям в этой весьма динамичной отрасли, обеспечивая тем самым ее постоянное совершенствование. В последней редакции руководства по TickIT активно используется идеология жизненного цикла программного обеспечения. Основные определения процессов жизненного цикла программного обеспечения даны в другом стандарте – ISO/IEC 12207, который может служить основой при выборе конкретной модели процессов жизненного цикла на предприятии. Таким образом, схема TickIT использует рациональное начало, заложенное в методах совершенствования процессов, таких популярных международных стандартов как CMM (SEI, USA), Trillium (Bell Canada), BOOTSTRAP и других.

13.8. Модели оценки зрелости бизнеса и технологических процессов в компаниях, разрабатывающих ПО

Концепция Baldrige Award

Объединение международных рынков, повышение требований к качеству и жесткая конкуренция привели к появлению двух параллельных концепций стандартизации качества – ISO 9000,

более распространенная в Европе, и Malcolm Baldrige National Quality Award (кратко Baldrige Award), весьма популярная в США. При этом ошибочно полагают, что обе концепции содержат примерно одни и те же требования, затрагивают одни и те же критерии качества и, следовательно, почти эквивалентны. Компании в соответствии с национальной принадлежностью или пристрастиями могут выбирать любую [10].

В действительности между двумя упомянутыми концепциями существует принципиальная разница в ориентации, целях и внутреннем содержании (по мнению экспертов, ISO 9000 перекрывает менее 10% критериев, вошедших в Baldrige Award). Главная цель Baldrige Award – научить организации создавать конкурентоспособные продукты. Основная цель ISO 9000 – заставить их жестко следовать описаниям технологических процессов, которые они сами же и составляют. Регистрация в ISO свидетельствует о том, что компания поддерживает «документированную практику». Однако качество конечного продукта, требования рынка и технологический уровень самих производственных процессов оказываются вне интересов модели ISO.

Сильная сторона концепции Baldrige Award состоит в том, что она ориентирована на виды деятельности, способствующие повышению конкурентоспособности компаний и предусматривающие для достижения этого различные способы: за счет обращения лицом к рынку и клиентам, нацеленности на конечный результат, постоянного совершенствования деловых процессов, тесной привязки к общей стратегии бизнеса, интеграции процессов на основе анализа, повышения квалификации персонала, расширения информационных связей.

При создании Baldrige Award преследовались три основные цели:

- донести до промышленных кругов важность идеи постоянного совершенствования качества технологических процессов;
- стимулировать компании к внедрению систем управления качеством;
- сделать всеобщим достоянием информацию о любых новых стратегиях и методиках, способствующих повышению качества.

Каждый год Malcolm Baldrige National Quality Award проводит награждение компаний, добившихся выдающихся успехов в области качества. Оценка строится по 28 критериям, разбитым на семь категорий:

1. Лидерство на рынке.
2. Информация и анализ.
3. Стратегия планирования качества.
4. Управление персоналом.
5. Управление качеством.
6. Достигнутые качественные результаты.
7. Мнения клиентов.

Существующие различия между ISO 9000 (2000) и Baldrige Award не означают, что одна система исключает другую. Наоборот, многие ведущие американские компании успешно их сочетают.

Модель для оценки зрелости возможностей компании, разрабатывающей программное обеспечение

В 1982 г. Министерство обороны США образовало комиссию, основной задачей которой стало исследование проблем, возникающих при разработке программных продуктов в организациях министерства. В результате деятельности комиссии в декабре 1984 г. был создан Институт инженеров-разработчиков программного обеспечения (Software Engineering Institute – SEI) на базе Университета Карнеги-Меллона в Нью-Йорке.

В 1986 г. SEI и корпорация Mitre под руководством Уоттса Хамфри (Watts Humphrey) приступили к разработке критериев оценки зрелости технологических процессов и компании в целом (Capability Maturity Model – CMM) [16-24].

В 1987 году SEI публикует: краткое описание структуры CMM; методы оценки процессов разработки ПО; методы оценки зрелости процессов производства ПО; анкету для выявления степени зрелости процессов (для проведения самоаудита и внешнего аудита).

- 1991 г. Выпуск версии CMM v1.0.
- 1992 г. Выпуск версии CMM v1.1.
- 1997 г. Выпуск очередной (усовершенствованной) версия CMM.

Методология CMM разрабатывалась и развивалась в США как средство, позволяющая выбирать наилучших производителей для выполнения госзаказов по разработке программного обеспечения. Для этого предполагалось создать критерии оценки зрелости ключевых процессов компании-разработчика и определить набор действий, необходимых для их дальнейшего совершенствования. В итоге методология оказалась чрезвычайно полезной для большинства компаний, стремящихся качественно улучшить существующие процессы проектирования, разработки, тестирования программных средств и свести управление ими

CMM де-факто стал именно таким стандартом. Его применение позволяет поставить разработку ПО на промышленную основу, повысить управляемость ключевых процессов и производственную культуру в целом, гарантировать качественную работу и исполнение проектов точно в срок. Основой для создания CMM стало базовое положение, что фундаментальная проблема «кризиса» процесса разработки качественного ПО заключается не в отсутствии новых методов и средств разработки, а в неспособности компании организовать технологические процессы и управлять ими.

Для оценки степени готовности предприятия разрабатывать качественный программный продукт CMM вводит ключевое понятие *зрелость организации* (Maturity). Незрелой считается организация, в которой:

- отсутствует долговременное и проектное планирование;
- процесс разработки программного обеспечения и его ключевые составляющие не идентифицированы, реализация процесса зависит от текущих условий, конкретных менеджеров и исполнителей;
- методы и процедуры не стандартизированы и не документированы;
- результат не предопределен реальными критериями, вытекающими из запланированных показателей, применения стандартных технологий и разработанных метрик;
- процесс выработки решения происходит стихийно, на грани искусства.

В этом случае велика вероятность появления неожиданных проблем, превышения бюджета или невыполнения сроков сдачи проекта. В такой компании, как правило, менеджеры и разработчики не управляют процессами – они вынуждены заниматься разрешением текущих и спонтанно возникающих проблем. Отметим, что на данном этапе развития находится большинство российских компаний.

Основные признаки зрелой организации:

- в компании имеются четко определенные и документированные процедуры управления требованиями, планирования проектной деятельности, управления конфигурацией, создания и тестирования программных продуктов, отработанные механизмы управления проектами;
- эти процедуры постоянно уточняются и совершенствуются;
- оценки времени, сложности и стоимости работ основываются на накопленном опыте, разработанных метриках и количественных показателях, что делает их достаточно точными;
- актуализированы внешние и созданы внутренние стандарты на ключевые процессы и процедуры;
- существуют обязательные для всех правила оформления методологической программной и пользовательской документации;
- технологии незначительно меняются от проекта к проекту на основании стабильных и проверенных подходов и методик;
- максимально используются наработанные в предыдущих проектах организационный и производственный опыт, программные модули, библиотеки программных средств;
- активно апробируются и внедряются новые технологии, производится оценка их эффективности.

CMM определяет *пять уровней* технологической зрелости компании, по которым заказчики могут оценивать потенциальных претендентов на получение контракта, а разработчики могут совершенствовать процессы создания ПО.

Каждый из уровней, кроме первого, состоит из нескольких *ключевых областей процесса* (Key Process Area), содержащих *цели* (Goals), *обязательства по выполнению* (Commitment to Perform), *осуществимость выполнения* (Ability to Perform), *выполняемые*

действия (Activity Performed), *их измерение и анализ* (Measurement and Analysis) и *проверку внедрения* (Verifying Implementation). Таким образом, CMM фактически является комплексом требований к ключевым параметрам эффективного стандартного процесса разработки ПО и способам его постоянного улучшения. Выполнение этих требований, в конечном счете, увеличивает возможности предприятия в достижении поставленных целей в области качества.

Начальный уровень (Initial Level – Level 1).

К данному уровню относится компания, которой удалось получить заказ, разработать и передать заказчику программный продукт. Стабильность разработок отсутствует. Лишь некоторые процессы определены, результат всецело зависит от усилий отдельных сотрудников. Успех одного проекта не гарантирует успешности следующего. К этой категории можно отнести любую компанию, которая хоть как-то исполняет взятые на себя обязательства. Ключевые области процесса этого уровня не зафиксированы.

Повторяемый уровень **(**Repeatable Level – Level 2).

Этому уровню соответствуют предприятия, обладающие определенными технологиями управления и разработки. Управление требованиями и планирование в большинстве случаев основываются на разработанной документированной политике и имеющемся опыте. Установлены и введены в повседневную практику базовые показатели для оценки параметров проекта. Менеджеры отслеживают выполнение работ и контролируют временные и производственные затраты.

В компании разработаны некоторые внутренние стандарты и организованы специальные группы проверки качества (QA). Изменения версий конечного программного продукта и созданных промежуточных программных средств отслеживаются в системе управления конфигурацией. Имеется необходимая дисциплина соблюдения установленных правил. Эффективные методики и процессы институционализируются (устанавливаются), что обеспечивает возможность повторения успеха предыдущих проектов в той же прикладной области. Ключевые области процесса разработки ПО этого уровня следующие:

1. Управление требованиями (Requirements management).
2. Планирование проекта разработки ПО (Software project planning).
3. Отслеживание хода проекта и контроль (Software project tracking and oversight).
4. Управление субподрядом разработки ПО (Software subcontract management).
5. Обеспечение качества разработки ПО (Software quality assurance).
6. Управление конфигурацией продукта (Software configuration management).

Определенный уровень (Defined Level – Level 3).

Уровень характеризуется детализированным методологическим подходом к управлению (то есть, описаны и закреплены в документированной политике типичные действия, необходимые для многократного повторения: роли и ответственность участников, стандартные процедуры и операции, порядок действий, количественные показатели и метрики процессов, форматы документов и пр.).

Для создания и поддержания методологий в актуальном состоянии в организации уже подготовлена и постоянно функционирует специальная группа. Компания регулярно проводит специальные тренинги для повышения профессионального уровня своих сотрудников.

Начиная с этого уровня, организация практически перестает зависеть от личностных качеств конкретных разработчиков и не имеет тенденции опускаться на нижестоящие уровни. Эта независимость обусловлена продуманным механизмом постановки задач, планирования мероприятий, выполнения операций и контроля исполнения.

Управленческие и инженерные процессы документированы, стандартизованы и интегрированы в унифицированную для всей организации технологию создания ПО. Каждый проект использует утвержденную версию этой технологии, адаптированную к особенностям текущего проекта. Ключевые области процесса разработки ПО этого уровня:

1. Настройка (стандартного) процесса организации (Organization Process Focus).
2. Определение (стандартного) процесса организации (Organization Process Defi-nition).
3. Программа обучения (Training Program).
4. Интегрированное управление разработкой ПО (Integrated Software Management).
5. Технология разработки ПО (Software Product Engineering).
6. Межгрупповая координация (Intergroup Coordination).
7. Экспертные (совместные) оценки (Peer Reviews).

Управляемый уровень (Managed Level – Level 4).

Уровень, при котором разработаны и закреплены в соответствующих нормативных документах количественные показатели качества. Более совершенное управление проектами достигается за счет уменьшения отклонений различных показателей проекта от запланированных. При этом систематические изменения в производительности процесса (тенденции, тренды) можно выделить из случайных вариаций (шума) на основании статистической обработки результатов измерений по процессам, особенно в хорошо освоенных и достаточно формализованных процессных областях. Ключевые области процесса разработки ПО этого уровня:

1. Количественное управление процессом (Quantitative Process Management).
2. Управление качеством разработки ПО (Software Quality Management).

Оптимизирующий уровень (Optimizing Level – Level 5).

Для этого уровня мероприятия по совершенствованию рассчитаны не только на существующие процессы, но и на внедрение, использование новых технологий и оценку их эффективности. Основной задачей всей организации на этом уровне является постоянное совершенствование существующих процессов, которое в идеале направлено на предотвращение известных ошибок или дефектов и предупреждение возможных. Применяется механизм повторного использования компонентов от проекта к проекту (шаблоны отчетов, форматы требований, процедуры и

стандартные операции, библиотеки модулей программных средств). Ключевые области процесса разработки ПО этого уровня:

1. Предотвращение дефектов (Defect Prevention).
2.Управление изменением технологий (Technology Change Management).
3. Управление изменением процесса (Process Change Management).

Всего CMM определяет следующий минимальный набор требований: реализовать18 ключевых областей процесса разработки ПО, содержащих 52 цели, 28 обязательств компании, 70 возможностей выполнения (гарантий компании) и 156 ключевых практик Рис.(13.8.1.).

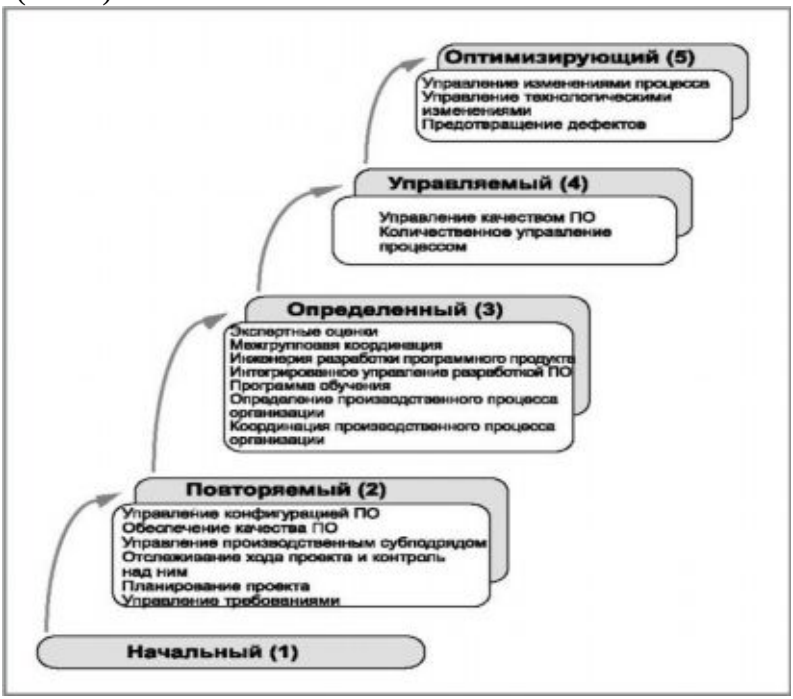


Рис. 13.8.1. Распределение ключевых областей процесса по уровням зрелости.

В результате аудита и аттестации компании присваивается определенный уровень, который при последующих аудитах в дальнейшем может повышаться или понижаться. Следует отметить,

что каждый следующий уровень в обязательном порядке включает в себя все ключевые характеристики предыдущих. В связи с этим сертификация компании по одному из уровней предполагает безусловное выполнение всех требований более низких уровней.

К преимуществам модели SEI SW-CMM относится то, что она ориентирована («заточена») на организации, занимающиеся разработкой программного обеспечения. В этой модели удалось более детально проработать требования, специфичные для процессов, связанных с разработкой ПО. Вследствие этого в SEI SW-CMM приведены не только требования к процессам организации, но и примеры реализации этих требований.

Основным же недостатком SW-CMM является то, что модель не авторизована в качестве стандарта ни международными, ни национальными органами по стандартизации. Вследствие этого применение и адаптация SW-CMM на национальном уровне объективно вызывает определенные трудности.

К недостаткам данной модели необходимо отнести также большие внешние накладные расходы на приведение процессов компании в соответствие модели CMM, нежели к моделям ISO 9000. Это связано с меньшей распространенностью модели в мире, меньшим количеством консалтинговых органов и экспертов и, в результате, с гораздо большими внешними затратами на консалтинг и на подтверждение соответствия процессов независимой третьей стороной.

Модель оценки Trillium

Модель Trillium, созданная в 1994 г. фирмами Bell Canada, Nothern Telecom и Bell-Nothern Research, предлагает еще один способ оценки процессов выпуска продуктов в телекоммуникационной и информационной областях и охватывает все аспекты жизненного цикла ПО. Хотя в ее основе и лежит рассмотренная выше модель CMM, эти модели существенно отличаются друг от друга. Помимо CMM, Trillium опирается на ряд других регламентирующих документов и стандартов: ISO 9001 и ISO 9000-3, стандарты Bellcore TR-NWT, значительную часть стандартов Malcolm Baldrige National Quality Award, стандарты IEEE и IEC [15.10].

Моделью Trillium охвачены следующие виды деятельности:

- проектирование бизнес-процессов (Business Process Engineering);
- создание сред разработки (Development Environment);
- совместное проектирование (Concurrent Engineering);
- надежное проектирование (Reliable Engineering);
- системное проектирование (System Engineering);
- управление качеством (Quality Management);
- оценка технологической зрелости (Technological Maturity Assessment);
- поддержка клиентов/партнерство (Customer Support/Partnership).

Интересен и способ классификации уровней зрелости в модели Trillium: в его основе лежит оценка фактора риска**.** Выделено пять уровней зрелости:

Уровень 1. Характеризуется хаотичностью. Качество продуктов низкое, сроки завершения проектов нарушаются. Риск высокий.

Уровень 2. Повторяемый и ориентированный на процессы. Успех проектов обусловлен внедрением систем управления проектами, планирования и менеджмента. Особое внимание уделяется выработке исходных требований, конфигурационному менеджменту и оценке качества готовых систем. Риск средний.

Уровень 3. Определенный и ориентированный на процессы. Все производственные процессы определены (т. е. зафиксированы) и используются в масштабе всей компании, хотя «подкрутка» отдельных процессов в целях успешного выполнения проектов допускается. Процессы полностью контролируются и постоянно совершенствуются. Стандарты ISO 9001 внедрены в части обучения персонала и внутреннего аудита. Риск невысокий.

Уровень 4. Управляемый и интегрированный. Основным средством повышения качества процессов становятся анализ и инструментальные системы. Функции отслеживания изменений и профилактики ошибок встраиваются в процессы. Активно используются CASE-средства. Риск довольно небольшой.

*Уровень 5***.** Полностью интегрированный. Широко применяются формализованные методологии. Для хранения истории разработки применяется репозиторий. Риск минимальный.

ISO15504 SPICE

Другой альтернативой CMM является модель SPICE (Software Process Improvement and Capability dEtermination) и ее развитие – международный стандарт ISO 15504 [15.25]. Он создан в ISO на основе CMM и при непосредственном участии того же SEI. Отличительной особенностью SPICE/ISO15504 по сравнению с CMM является уход от последовательно-ступенчатого (Staged) представления процесса разработки ПО и самого понятия технологической зрелости всей компании по уровням (Maturity Level).

В 1991 году Международная организация по стандартизации инициировала работу по созданию единого стандарта оценки процессов компании, разрабатывающей программное обеспечение. При этом речь идет о возможности оценки как всей совокупности процессов компании, так и отдельных «продвинутых» процессов.

Официально стандарт называется «ISO/IEC 15504: Information Technology – Software Process Assessment», но его часто называют SPICE (**S**oftware **P**rocess **I**mprovement and **C**apability d**E**termination – «Определение возможностей и улучшение процесса создания программного обеспечения»).

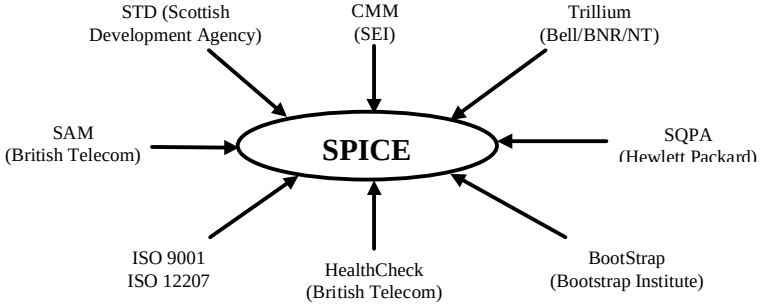


Рис. 13.8.2. Источники для составления стандарта SPICE.

Главной задачей было создание международного стандарта, в котором был бы учтен весь накопленный опыт в области разработки ПО. На Рис. 13.8.2 показаны наиболее значительные стандарты, идеи и материалы которых были использованы при создании SPICE. Как и в CMM, основной задачей организации по модели

SPICE является *постоянное улучшение процесса разработки ПО*. Кроме того, в SPICE тоже используется схема с различными уровнями возможностей (в SPICE определено шесть различных уровней), но эти уровни применяются не только к организации в целом, но и к отдельно взятым процессам.

В Табл. 13.3 приведен список «уровней возможностей» модели SPICE и характерные для них процедуры управления (на данный момент не существует русского перевода стандарта SPICE, поэтому использованные термины не являются общепринятыми или официально зарегистрированными).

Таблица 13.3. Уровни возможностей процесса в стандарте SPICE.

Уровни	**Название**
Уровень 0	Процесс не выполняется
Уровень 1	Выполняемый процесс
1.1	Измерение производительности процесса
Уровень 2	Управляемый процесс
2.1	Управление производительностью
2.2	Управление созданием продуктов
Уровень 3	Установленный процесс
3.1	Документирование процесса
3.2	Отслеживание ресурсов процесса
Уровень 4	Предсказуемый процесс
4.1	Измерение процесса
4.2	Управление процессом
Уровень 5	Оптимизирующий процесс
5.1	Изменение процесса
5.2	Постоянное совершенствование

Таким образом, процессы в модели SPICE могут быть представлены в виде как на Рис. 13.8.3.

Видно, что модель SPICE является двумерной – на одной оси откладываются процессы, а на другой оси – достигнутый уровень возможностей для этих процессов.

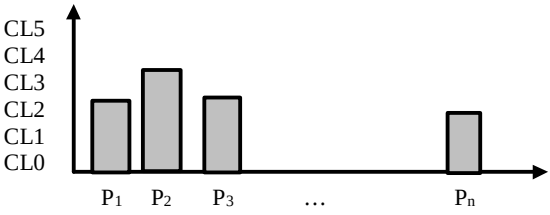


Рис. 13.8.3. Упрощенная модель оценки процессов в стандарте SPICE.

1. *Оценка процесса* происходит путем сравнения процесса разработки ПО, существующего в данной организации, с описанной в стандарте моделью. Анализ результатов, полученных на этом этапе, помогает определить сильные и слабые стороны процесса, а также внутренние риски, присущие данному процессу, помогает оценить эффективность процессов, определить причины ухудшения качества и связанные с этим издержки во времени или стоимости.
2. *Определение возможностей процесса* позволяет оценить возможности улучшения данного процесса. Очень часто определение возможностей процесса производится компанией-поставщиком, чтобы убедить существующих или потенциальных заказчиков в своей способности достичь заданных показателей.
3. В результате предыдущих шагов, в организации может появиться понимание необходимости улучшения того или иного процесса. К этому моменту цели совершенствования процесса уже четко сформулированы и остается только техническая реализация поставленных задач. После этого весь цикл работ начинается сначала.

В стандарте выделены пять основных категорий процессов:

- потребитель – поставщик (Customer – Supplier, CUS)
- инженерная (Engineering, ENG)
- Вспомогательная (Support, SUP)
- Управленческая (Management, MAN)
- Организационная (Organization, ORG)

Набор документов по SPICE состоит из девяти частей (Рис. 13.8.4). Первая часть "Введение и основные концепции" и девятая часть "Словарь" носят чисто информативный характер. Самым важным элементом SPICE является оценка и аудит процессов,

поэтому ей посвящена наибольшая часть документов, а именно части со второй по шестую. Например, вторая часть стандарта содержит так называемую "эталонную модель" (reference model), которая описывает процессы. Практически, это модель процессов из ISO 12297, хотя, к сожалению, эти модели не полностью идентичны. Остальные части стандарта – седьмая и восьмая – посвящены соответственно улучшению процесса, и определению возможностей процесса.

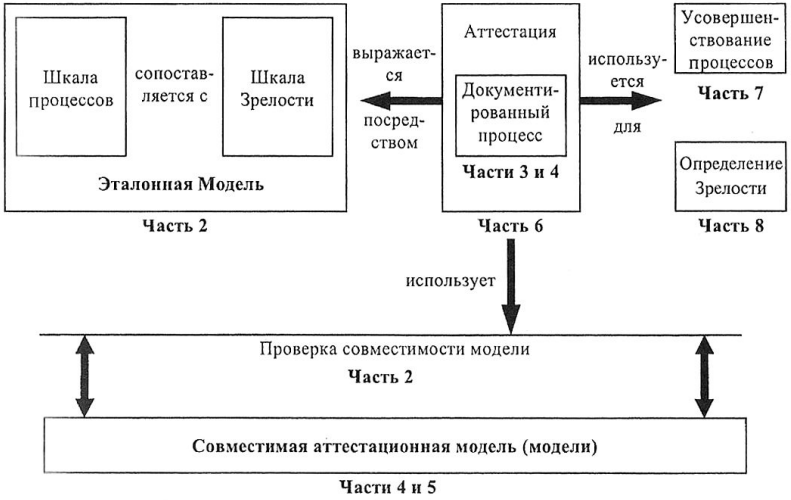


Рис. 13.8.4. Совместимость частей стандарта ISO 15504.

Одним из важных достоинств SPICE является его открытость и свободное распространение. В Табл. 13.4 кратко сформулированы преимущества SPICE по сравнению с ISO 9001.

Таблица 13.4 Сравнение стандартов SPICE и ISO 9001.

SPICE (ISO15504)	ISO 9001
Объемный и подробный документ	Краткий документ
Детальная модель	Абстрактная модель

Улучшение процесса и определение его возможностей	Только сертификация
Шесть уровней возможностей процессов	Сертификация/отказ
Конкретные требования к оценке процесса, руководство по применению	Модель качества в общем виде
Дополняет ISO 9001	Может быть детализирован с помощью SPICE

SPICE предоставляет более полный набор средств по обеспечению качества и улучшению процессов, чем ISO 9001 [26]. Поэтому для обеспечения качества процессов разработки ПО можно рекомендовать использовать именно SPICE. Это поможет организации заметно улучшить существующие процессы, а затем при необходимости получить и сертификат ISO 9001.

Отметим, что модель SPICE/ISO15504 (как и ISO 9000) не выделяет ключевых областей процесса разработки ПО, а лишь устанавливает общие и индивидуальные показатели уровня совершенства для любых процессов. Организации, которая использует SPICE/ISO 15504, по всей видимости, все-таки придется обратиться к CMM за критериями в формирования единого стандартного процесса разработки ПО и определения приоритетов своего развития.

Информационная модель процесса разработки ПО SCOPE*PROCEPT

В настоящее время Германский национальный исследовательский центр по информационным технологиям (GMD) завершает работу над проектом SCOPE*PROCEPT, охватывающим информационные модели процессов создания ПО, включая методы оценки качества на основе метрик. Уже сейчас модель SCOPE*PROCEPT опробована в 20 крупных промышленных проектах и внесена на обсуждение в ISO [15.10].

В состав SCOPE*PROCEPT входит несколько компонентов, соответствующих различным сторонам деятельности по созданию программ:

- специфицирование процессов программирования и тестирования (ProcePT). Данный компонент предназначен для выработки спецификаций и тестирования используемых в проектах технологических процессов. Разработка процессов ведется по нисходящей схеме, начиная с уровня общей модели процесса. Далее путем итерационных уточнений автоматически создается комплект необходимых документов: разнообразные справочники по проекту, описания типов деятельности, перечни продуктов, которые будут выпущены в ходе работы над проектом, соответствующие диаграммы и схемы;
- инжиниринг моделей качества (Model Y7). Суть его в определении модели качества для данного проекта с использованием методов оценки качества; подразумевает встраивание этой модели в существующие технологические и бизнес-процессы;
- измерение характеристик (метрик) ПО (SMV). Можно представить в виде своеобразного испытательного стенда, на котором проводятся количественные исследования характеристик программ; - интегрированная среда для измерения характеристик компиляторов (CISM);
- моделирование процессов производства ПО (SPM). Данный компонент предназначен для оценки качества процессов разработки, причем на основе количественных показателей.

Модель CMMI (Capability Maturity Model Integration)

Наконец, еще одна альтернатива CMM for Software – новая интегрированная модель технологической зрелости CMMI [27, 28]. CMMI (версия 1.1) появился в марте 2002 года. Эта модель является результатом усилий по интеграции созданных ранее моделей семейства CMM.

Целью разработки CMMI явилось желание его создателей (Питтсбургское отделение SEI) избежать проблем, связанных с использованием различных моделей CMM. Начиная с1991 года, были разработаны модели CMM для различных областей применения, наиболее существенные из них:

- Software (SW) CMM для организаций-разработчиков ПО;
- System Engineering (SE) CMM для организаций-разработчиков

систем (Electronic Industries Alliance Interim Standard – EIA/IS 731);

- Integrated Product Development (IPD) CMM для организаций-разработчиков интегрированных продуктов и технологий;
- Software Acquisition (SA) CMM для организаций-заказчиков ПО.

На основе этих моделей и был построен CMMI. Он вобрал в себя лучшее из этих моделей, устранив неоднозначность толкования некоторых понятий ввиду наличия множества моделей.

Модель CMMI является ответом на стандарт ISO 15504, гармонизирована с ним и предоставляет пользователю на выбор два варианта представления: последовательно-ступенчатое (как в SW CMM) и сплошное (как в SPICE/ISO 15504). CMMI является референтной моделью, которая шаг за шагом помогает организации усовершенствовать свои бизнес процессы.

Использование этой модели позволяет организации оценить эффективность бизнес-процессов, установить приоритетные направления их усовершенствования, а также внедрить данные усовершенствования. Однако следует помнить, что нельзя улучшать бизнес-процессы во имя их улучшения, данные улучшения должны помогать бизнесу, достичь поставленных перед ним целей. Также необходимо иметь в виду, что улучшение процессов это долговременное, стратегическое усилие организации.

Существует два подхода (репрезентации) в совершенствовании бизнес-процессов в контексте CMMI: непрерывная репрезентация и поэтапная репрезентация. При выборе непрерывной репрезентации организация оставляет за собой право выбора последовательности действий ведущих к совершенствованию бизнес процессов. В данном случае усовершенствуются процессы определенной области процессов. Данный подход позволяет мигрировать с модели EIA/IS 731 (если она уже применялась) на модель CMMI.

Поэтапная репрезентация предполагает определенную, доказавшую право на существование, последовательность действий, которая ведет к совершенствованию всех процессов организации в целом, а не определенной области процессов как в предыдущем подходе. Данная репрезентация помогает осуществить переход с модели SW-CMM к модели CMMI. Следует отметить, что наличие предыдущих моделей помогает перейти к модели CMMI, но ни в

коей мере не является необходимым условием для внедрения CMMI.

В непрерывной репрезентации для оценки (измерения) степени улучшения процессов используется *уровень устойчивости* (capability level), в то время как в поэтапной репрезентации используется *уровень зрелости* (maturity level). Основное различие между этими двумя понятиями заключается в следующем:

Непрерывная репрезентация. Уровни устойчивости, используемые в непрерывной репрезентации, применяются для улучшения процессов в каждой области процессов. Существует шесть таких уровней, пронумерованных от 0 до 5 (Табл.13.5). Уровень устойчивости включает в себя общую цель и набор общих и специфических практик (см. глоссарий). Непрерывная репрезентация имеет два типа специфических практик: общие и дополнительные, в поэтапной репрезентации такого деления нет.

Таблица 13.5. Уровни устойчивости процесса в соответствии с ISO 15504.

Уровень устойчивости	**Название уровня**
0	Незавершенный уровень
1	Выполненный уровень
2	Управляемый уровень
3	Определенный уровень
4	Количественно-управляемый уровень
5	Оптимизированный уровень

Поэтапная репрезентация. В свою очередь уровень зрелости описывает общую организационную зрелость, и он включает в себя предопределенный набор областей процессов. Существует пять уровней зрелости, пронумерованных от 1 до 5 (Табл.13.6). В поэтапной репрезентации может присутствовать лишь одна общая цель для одной области процессов.

Таблица 13.6. Уровни зрелости процесса в соответствии с ISO 15504.

Уровень зрелости	**Название уровня**
1	Начальный уровень
2	Управляемый уровень
3	Определенный уровень
4	Количественно-управляемый уровень
5	Оптимизированный уровень

Рис. 13.8.5 и 13.8.6 иллюстрируют разницу в этих двух подходах:

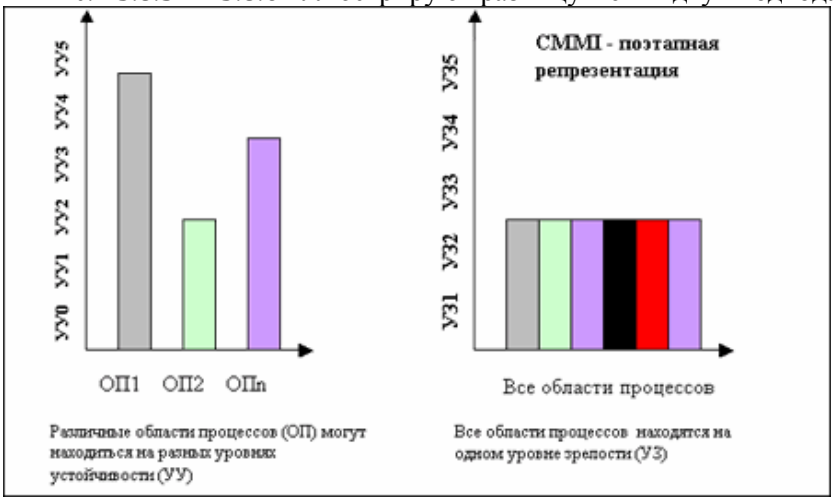


Рис. 13.8.5. Репрезентации в CMMI.

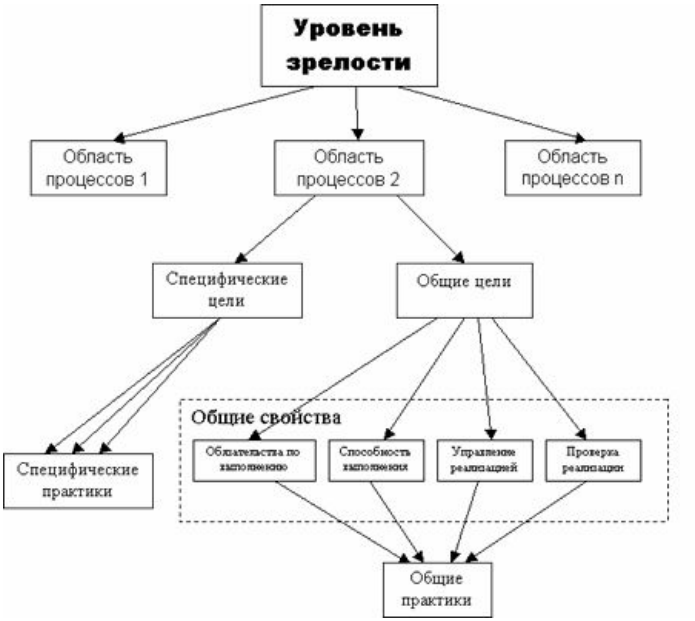


Рис. 13.8.6. Структурная схема CMMI – поэтапная репрезентация.

В Табл. 13.7 представлены уровни зрелости и соответствующие им области процесса (в случае использования поэтапной репрезентации предполагается, что любая организация по умолчанию находится на первом уровне модели зрелости процессов).

Таблица 13.7. Уровни зрелости и ключевые области процесса.

Уровень зрелости	Ключевая область процесса	Название	Цель
2	Менеджмент требований (Requirements Management)	REQM	Управление требованиями предъявляемым к продуктам проекта или компонентам продукта, с целью выявления несоответствия между требованиями и планами проекта.
	Планирование проекта (Project Planning)	PP	Разработка и поддержание планов определяющих развитие проекта
	Мониторинг и контроль проекта (Project Monitoring and Control)	PMC	Обеспечить понимание стадии разработки проекта с целью принятия корректирующих действий в случае серьезного отклонения от плана
	Менеджмент договоров с поставщиками (Supplier Agreement Management	SAM	Управление приобретением товаров и услуг от внешних поставщиков, с которыми заключены договоры
	Измерение и анализ (Measurement and Analysis)	M&A	Разработка и поддержание возможности измерения, используемой для поддержки нужд информационного менеджмента
	Оценка (гарантирование) качества товаров и процессов	PPQA	Обеспечение поддержки и управления в соответствии с целями процессов и связанными с ними продуктами работы

	(Process and Product Quality Assurance)		
	Конфигурационный менеджмент (Configuration Management)	CM	Установка и поддержание целостности продуктов работы (work products) в результате использования идентификации конфигураций, конфигурационного контроля и конфигурационного аудита
3	Разработка требований (Requirements Development)	RD	Сбор и анализ требований потребителей к продуктам и компонентам продуктов
	Техническое решение (Technical Solution)	TS	Разработка, дизайн и внедрение решений по согласованным требованиям. Решения, дизайн и внедрения выражены продуктами, компонентами продуктов и связанными с данными продуктами процессами.
	Интеграция продукта (Product Integration)	PI	Сборка (монтирование) продукта из его составляющих, проверка качества интеграции, ее функциональности и выпуск продукта
	Верификация (Verification)	Ver	Гарантирование того, что выбранные продукты работы отвечают предъявляемым требованиям
	Валидация (Validation)	Val	Демонстрация того, что продукт и его компоненты соответствуют его предполагаемому использованию в предполагаемой среде.
	Фокусирование на процессах организации (Organization Process Focus)	OPF	Установление и поддержание понимания процессов организации и процессных активов, идентификация, планирование и внедрение

			улучшений связанных с данными областями.
	Описание процессов организации (Organization Process Definition)	OPD	Установление и поддержание возможного к использованию массива процессов организации
	Организационный тренинг (Organizational Training)	OT	Повышение знаний и способностей людей для выполнения ими своих ролей эффективно и рационально
	Менеджмент интеграции проектов (Integrated Project Management)	IPM	Установка и управление проектом и вовлечение всех заинтересованных лиц в интегрированный и определенный процесс. Данная область также затрагивает общее видение проекта командой разработчиков
	Менеджмент рисков (Risk Management)	RSKM	Определение потенциальных проблем до их появления. В связи с этим процессы по снижению рисков могут планироваться и осуществляться на любом этапе разработки продукта или процесса.
	Интегрированные команды (разработчиков) Integrated Teaming	IT	Формирование и поддержание интегрированных команд для разработки продуктов работы (Work Products)
	Интегрированное управление поставщиками (Integrated Supplier Management)	ISM	Мониторинг новых продуктов, оценка источников продуктов, которые могут удовлетворить требованиям к проекту и использование данной информации для выбора поставщиков
	Анализ решений и разрешение (Decision Analysis and Resolution	DAR	Разработка решений на основе структурированного подхода, который позволяет оценить альтернативные решения на

			основе установленных критериев
	Организационная среда для интеграции (Organizational Environment for Integration)	OEI	Предоставление инфраструктуры для интегрированной разработки продуктов и процессов и управление людьми (персоналом) в целях интеграции
4	Производительный организационный процесс (Organizational Process Performance)	OPP	Установление и поддержание количественного понимания производительности набора стандартизированных процессов организации и обеспечение информацией о производительности процессов и моделей для количественного управления проектами организации.
	Количественный менеджмент проекта (Quantitative Project Management)	QPM	Количественно управлять определенным процессом в целях достижения установленного в рамках проекта качества и целей производительности.
5	Организационные инновации и внедрение (Organiza-tional Innovation and Deployment)	OID	Выбор и внедрение инноваций и улучшений, которые измеряемо, улучшают организационные процессы и технологии.
	Анализ причин и разрешение (Causal Analysis and Resolution)	CAR	Идентификация причин дефектов и других проблем и принятие действий предотвращающих их появление в будущем

Чем же отличаются процессы, находящиеся на разных уровнях модели зрелости? В нашем рассмотрении мы предположим использование поэтапной репрезентации. Как было упомянуто выше, в этом случае модель CMMI имеет пять уровней зрелости,

предполагается, что любая организация находится на первом уровне зрелости.

Процессы первого уровня зрелости, характеризуются хаотичностью, реактивностью, непредсказуемостью. Несмотря на это, очень часто организации, находящиеся на данном этапе развития, производят довольно качественные продукты. При этом, как правило, превышается бюджет и время разработки данных продуктов. Качественные продукты данных организаций производятся не за счет устойчивых и отлаженных процессов, а благодаря титаническим усилиям отдельных личностей.

В случае ухода таких людей очень тяжело повторить успешные проекты. На этом этапе очень тяжело предсказать производительность процессов, протекающих в организации.

На уровне 1 производственный процесс (а вместе с ним и все процессы) представляется аморфной сущностью, практически черным ящиком, представление о процессах очень ограниченное, чрезмерно много усилий тратится на выяснение статуса развития проекта и текущего хода работ (Рис. 13.8.7).

Рис. 13.8.7.

Уровень зрелости 2 – управляемый уровень. На данном этапе основные процессы описаны, их, возможно, использовать неоднократно. Другими словами, проекты, выполняемые организацией, отвечают требованиям. Процессы управляемы, они планируются, выполняются, измеряются и контролируются. Однако процессы все же имеют некоторую долю реактивности в своей сущности.

На уровне 2 контролируются требования заказчиков и промежуточные продукты, а также установлены основные практики управления проектом. Эти средства позволяют управлять проектом, однако дают фрагментарное представление о нем. Фактически, производственный процесс можно представить последовательностью черных ящиков и реальное видение проекта присутствует лишь на промежуточных этапах (Рис. 13.8.8).

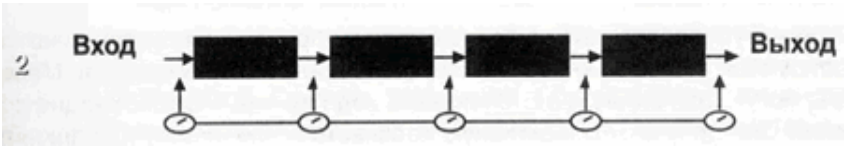


Рис. 13.8.8.

Уровень зрелости 3 – определенный уровень. В этом случае процессы определены. Установлены стандарты в пределах организации. На данном этапе процессы описаны не на уровне отдельного проекта, а на уровне всей организации. Присутствует более детальное описание всех процессов, в котором лучше раскрываются связи и зависимости, знание которых позволяет улучшить управление.

На этом уровне – уровне 3 - становится видимой внутренняя сторона наших черных ящиков. Это внутренняя структура отражает способ, применения стандартного производственного процесса организации (Рис. 13.8.9).

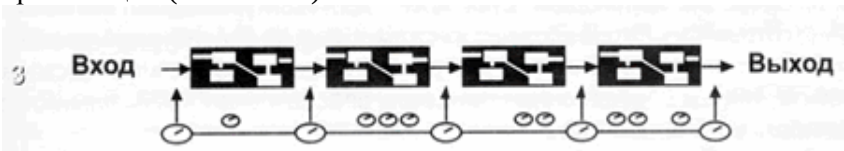


Рис. 13.8.9.

Уровень зрелости 4 – количественно-управляемый уровень. На данном этапе достигнуты все цели предыдущих уровней. Выбраны субпрактики, которые при использовании статистических методов и других количественных техник позволяют контролировать качество выполнения процессов. Самое главное отличие этого этапа от предыдущего заключается в предсказуемости эффективности процессов и возможности ею (эффективностью) управлять.

На уровне 4 определенные процессы количественно контролируются с помощью соответствующих средств и техник (Рис. 13.8.10).

Уровень зрелости 5 – уровень постоянного улучшения (оптимизации) процессов. На данном этапе мы имеем точные характеристики оценки эффективности бизнес процессов, что позволяет нам постоянно и эффективно улучшать бизнес процессы

путем развития существующих методов и техник и внедрения новых (Рис. 13.8.11).

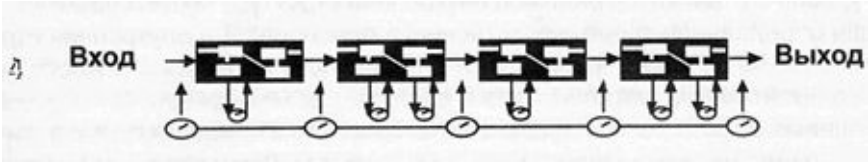


Рис. 13.8.10.

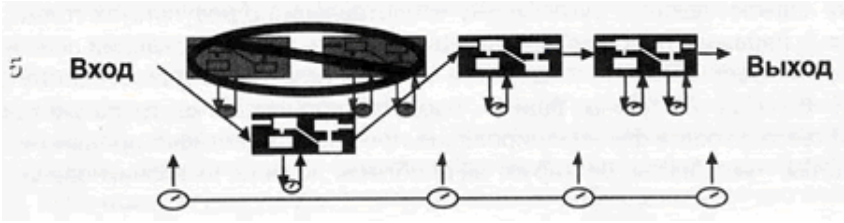


Рис. 13.8.11.

13.9. Общее управление качеством при разработке ПО

Управление проектами – это приложение знаний, опыта, методов и средств для удовлетворения требований, предъявляемых к проекту, и ожиданий участников проекта. Чтобы удовлетворить эти требования и ожидания, необходимо найти оптимальное сочетание между целями, задачами, сроками, затратами, реализацией качества и другими характеристиками проекта.

Одним из важных моментов, который необходимо иметь в виду при внедрении каких-либо стандартов (ISO 9000, SEI SW-CMM, TickIT, Spice ISO 15504 и т.п.), связан с тем, что структура производства компаний, разрабатывающих программное обеспечение, связана со спецификой продукта. Часто продукт, разрабатываемый IT-компанией, уникален. И для его разработки, как правило, используется проектный тип

организации производства, который тесно связан с матричной структурой управления проектами.

Комитет ISO № 176 разработал рекомендательный стандарт ISO 10006 "Менеджмент качества. Руководство качеством при управлении проектами", который определяет основные подходы к управлению проектами и определяет его место в модели обеспечения качеством. Авторы стандартов ISO серии 9000 определяют процесс управления проектами как часть системы менеджмента качества. С другой стороны, возможен и противоположный взгляд (которого придерживаются оппоненты стандартов ISO серии 9000), согласно которому менеджмент качества является одной из составной частей системы управления проектами.

Управление проектами является неотъемлемой частью управления качеством в организациях разработчиков программного обеспечения. Поэтому неудивительно, что для приведения в соответствие системы управления качеством производства к требованиям модели ISO 9001 и к требованиям модели улучшения процессов производства SEI SW-CMM использование стандартов и признанных в мире технологий по управлению проектами является краеугольным камнем развития внутренних технологий в IT-компаниях.

Взаимосвязь наиболее признанных и применяемых в мире стандартов в области разработки программного обеспечения представлена на Рисунке 13.9.1. Видим, что они не покрываю полностью требуемую область TQM. Отсюда вывод, что для полной реализации требований TQM необходимо на базе указанных стандартов разрабатывать свою систему корпоративных стандартов.

Посмотрим, как соотносятся между собой рассмотренные модели.

Основное сходство моделей ISO 9000 и SEI SW-CMM заключается в том, что в их основу положена единая теория TQM, основанная на поэтапном улучшении внутренних производственных процессов за счет множества небольших, но

постоянно и последовательно внедряемых в компании улучшений (принцип «Kaizen»), и удовлетворения всех заинтересованных в функционировании организаций сторон (клиент, государство, персонал компании, акционеры компании, субподрядчики и т.д.).

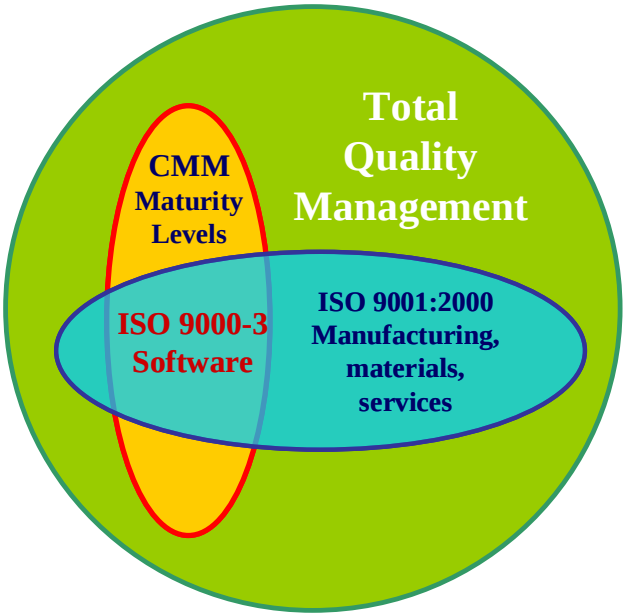


Рис. 13.9.1. Взаимосвязь и области взаимного покрытия стандартов.

Несмотря на то, что методология, взятая за основу обеих моделей в целом совпадает, подходы к построению самосовершенствующихся систем управления качеством и к улучшению производственных процессов, которые в них определены, отличаются. Самое главное отличие модели SEI SW-CMM от моделей ISO 9000 заключается в том, что модель CMM в основном ориентирована на *построение системы постоянного улучшения процессов для повышения уровня зрелости компании*, в то время как модели на основе ISO 9000 – *на построение элементной системы качества*, в которой отсутствует четкая методология оценки и постоянного совершенствования процессов.

Помимо этого, в отличие от моделей ISO 9000, где (для того чтобы им соответствовать и получить сертификат) необходимо продемонстрировать единовременное 100% соответствие всем требованиям модели, в модели SEI SW-CMM предусмотрен плавный, поэтапный подход к построению системы совершенствования процессов: можно поэтапно получать независимые подтверждения об улучшении процессов после каждого уровня зрелости.

Такой поэтапный подход положительно сказывается на результатах внедрения модели. Желание получить сертификат соответствия в самые короткие сроки вынуждает консалтинговые компании и специалистов, занимающихся управлением качества, использовать гибкость высокоуровневых моделей в «корыстных» целях. В результате такого форсирования событий у организации, получившей сертификат по ISO 9000:

- определен только минимально-необходимый набор процессов для соответствия ISO 9001, а не все процессы, которые требуются компании для эффективного функционирования;
- уровень детализации процессов не достаточен для четкого понимания того, что творится внутри процессов и кто, за какие задачи внутри процесса отвечает;
- недостаточно «протестированные» временем процессы. В лучшем случае через новые требования прошли лишь несколько пилотных проектов и через какое-то время становятся ясным необходимость их корректировки и дополнения. Часто, сразу после сертификации о построенной системе качества и процессах забывают до следующего надзорного аудита, забывая при этом и о затраченных финансовых ресурсах и энтузиазме сотрудников.

Действительно, когда выступаешь в роли независимого аудитора, очень сложно доказать, что принятый уровень детализации процесса явно недостаточен для эффективного функционирования системы качества компании. Но и доказать обратное за время, которое выделяется на аудит по ISO 9000, крайне сложно.

Практика показывает, что быстро построить эффективные процессы 5-го уровня зрелости (также как и процессы на основе

модели ISO 9000) невозможно. Для того чтобы этого добиться, недостаточно просто описать процессы с учетом требований модели.

Самая главная сложность заключается в том, что необходимо перепроектировать культуру производства внутри организации. И сделать это волевым решением руководства невозможно. Именно поэтому подход, который определен в модели SEI SW-CMM, более жизнеспособен и реалистичен, чем в моделях ISO 9000.

И все же, учитывая все «за» и «против», какой подход развития внутренних процессов наиболее целесообразно выбрать Российским компаниям, занимающимся разработкой ПО?

Конечно же, это зависит от множества обстоятельств:

- от уровня развития существующих в компании технологий;
- от имеющегося времени у организации для получения независимого подтверждения соответствия процессов выбранной модели;
- от начальных инвестиций;
- от расстановки приоритетов в получении выгоды от инвестиций в совершенствование процессов;
- от региона, на который планирует ориентировать продажу услуг компания.

Но если исходить из соотношения «трудозатраты / эффективность / результат», то, как показывает практика, наиболее оправданный путь заключается в следующей стратегии (с экономической, методологической и практической точек зрения):

1.Привести существующие процессы в соответствия с требованиями модели ISO 9001:2000.

2.Начать внедрять SW-CMM, используя эту модель для улучшения существующих процессов с дальнейшим проведением оценки (на данном этапе рекомендуется наблюдать за развитием моделей ISO 15504 SPICE и SEI-CMMI)

Этот подход рационален по нескольким причинам.

Во-первых, как сказано выше, сертификация по модели ISO 9000 более известна в мире и имеет широкое применение. Вследствие этого большинству Европейских заказчиков сертификат по модели ISO 9001:2000 будет служить достаточным основанием для начала разговора о возможном сотрудничестве. Необходимо также

заметить, что стандарты ISO серии 9000 признаны в качестве национальных более чем в 100 странах мира, и Россия не является исключением. Существуют официальные переводы стандартов на русский язык, и нет проблем, связанных с подготовкой специалистов.

Во-вторых, затраты компании на привлечение консультантов и аудиторов по моделям ISO 9000 на порядок меньше стоимости экспертов по модели SEI SW-CMM. Не существует языкового барьера. Многие ведущие организации, занимающиеся сертификацией по моделям ISO 9000, имеют свои представительства в России (BVQI, DNV, TUV-CERT, LRQA), вследствие чего полностью реализована потребность в русскоязычных аудиторах.

Для сравнения: по модели SEI SW-CMM зарегистрировано SEI около 300 человек, имеющих право возглавлять команду по оценке зрелости процессов на соответствие модели SEI SW-CMM. К сожалению, среди них пока нет ни одного русскоязычного эксперта.

В-третьих, внедрив требования модели ISO 9001:2000, организация на 80-90% закроет требования к процессам второго уровня зрелости, и создаст необходимые предпосылки для внедрения требований третьего и выше уровня зрелости процессов по модели SEI SW-CMM.

По оценкам западных IT-компаний для организации выгоднее находиться на третьем и выше, либо на первом уровне зрелости, нежели на втором. Второй уровень зрелости несколько замедляет процессы разработки из-за большого количества документов, создаваемых по проектам. По оценке наших экспертов организация, чьи процессы соответствуют модели ISO 9001:2000, находятся на более высоком уровне развития, нежели второй уровень зрелости SEI SW-CMM, и имеют множество реализованных требований из 3-го, 4-го и даже большую часть требований пятого уровня зрелости процессов CMM.

И последнее, но не менее важное, преимущество данного подхода: по оценкам наших экспертов, организация зачастую быстрее достигает третьего уровня зрелости процессов по SEI SW-CMM, проходя через сертификацию по ISO 9001:2000, чем

напрямую занимающиеся совершенствованием процессов по CMM, Level 3.

Основная сложность внедрения современных моделей менеджмента качества заключается не в технической стороне вопроса, а в изменении мышления сотрудников. Внедрение модели ISO 9001:2000 подготавливает благоприятную почву для дальнейшего развития процессов.

Установление единого стандартного процесса разработки программного обеспечения в масштабах компании в соответствии с требованиями CMM

Цель – реальное достижение компанией 3-го уровня зрелости в соответствии с требованиями стандарта CMM и последующая сертификация SEI на этот уровень.

Задача – создание и установление стандартного процесса организации (СПО) по разработке программного обеспечения, включающего управленческие, производственные и ресурсные составляющие.

Необходимость – компания вышла на уровень работы по проектам, когда установление стандартного процесса разработки ПО *стало насущной необходимостью* для большей части сотрудников компании.

Проект – разработка и установление СПО является реальным проектом, выполняемым по единому плану работ за запланированный период с использованием запланированных ресурсов.

Осуществимость – в компании есть необходимые предпосылки достижения поставленной цели:

- накопленные материалы и опыт выполненных проектов;
- зрелые специалисты в области проектирования, специфицирования, разработки и тестирования программного обеспечения;
- сотрудники, прошедшие обучение, в том числе и по стандарту CMM;
- начальные методологические разработки;

- достаточная материально-техническая и инструментальная база (REAL) для проектирования и разработки элементов стандартного процесса.

Команда – работа выполняется двумя основными группами:

- *группой разработки единого процесса*, которая разрабатывает элементы СПО, документированные политики в соответствии с требованиями ключевых областей процесса, проводит стандартизацию процесса, разрабатывает методологии его применения, оптимизации и совершенствования;
- *группой проектной СММ-поддержки и сопровождения*, которая решает вопросы настройки основного процесса на конкретные проекты и их сопровождения в соответствии с требованиями СММ, разрабатывает текущие методологии, проектные метрики и базы данных, пополняет межпроектную БД.

Группа разработки СПО работает на постоянной основе, состав *группы проектного СММ-сопровождения* может изменяться в зависимости от требований проекта. Группы работают по согласованным планам, черпая необходимые материалы из текущих проектов.

Руководство и контроль – ввиду особой важности проекта необходим постоянный контроль и участие представителей руководства компании в работе групп.

Предлагается участие двух представителей руководства: работой *группы разработки СПО* руководит и контролирует заместитель генерального директора по качеству, *работой группы проектной СММ-поддержки* – исполнительный директор (менеджер) проекта.

Аудит – в ходе начальной фазы работ по проекту необходимо провести аудит выполненных проектов для выявления наработанных материалов, которые можно использовать при разработке элементов СПО, проектного СММ-сопровождения проектов и создания межпроектной базы данных.

Обучение – для запуска проекта следует провести начальное обучение членов групп для знакомства со структурой СММ, целями и требованиями ключевых областей процесса. Последующее текущее обучение можно проводить в ходе разработок и выполнения работ по проектам.

Сроки – достижение 3-го уровня предполагает реализацию

требований 13 ключевых областей процесса:

1. Управление требованиями (Requirements management)
2. Планирование проекта разработки ПО (Software project planning)
3. Отслеживание хода проекта и контроль (Software project tracking and oversight)
4. Управление субподрядом разработки ПО (Software subcontract management)
5. Обеспечение качества разработки ПО (Software quality assurance)
6. Управление конфигурацией продукта (Software configuration management)
7. Настройка [стандартного] процесса организации (Organization Process Focus)
8. Определение [стандартного] процесса организации (Organization Process Definition)
9. Программа обучения (Traning Program)
10. Интегрированное управление разработкой ПО (Integrated Software Management)
11. Технология разработки ПО (Software Product Engineering)
12. Межгрупповая координация (Intergroup Coordination)
13. Экспертные оценки (Peer Reviews).

Эти ключевые области содержат в общей сложности 37 целей, 18 обязательств, 49 предпосылок для выполнения, гарантируемых возможностями компании, а также **112 ключевых практик**, которые необходимо разработать и внедрить *в постоянное использование.*

Предполагается параллельная работа по всем ключевым областям. В этом случае минимальная реализация процесса для текущих нужд компании может занять 6-8 месяцев, подготовка к реальной сертификации – в лучшем случае, до года.

Всего CMM определяет следующий минимальный набор требований: 52 цели, 28 обязательств компании, 70 возможностей выполнения (гарантий компании) и 150 ключевых практик.

Ресурсы – обеспечение работы двух непосредственных CMM-групп, закупка или подписка на необходимые стандарты SEI и IEEE, перевод части документов с английского на русский и с

русского на английский языки (для внешнего аудита и сертификации). Величина финансового сопровождения проекта определяется в ходе предварительного аудита после оценки объема, сложности и продолжительности работ и состава СММ-групп.

Ответственность – ответственность за реализацию проекта в целом возлагается персонально на указанных представителей руководства, у которых имеются достаточные полномочия.

Ответственность за текущие работы по СПО и СММ-поддержке проектов персонально распределяется по руководителям департаментов и отделов, которые в обязательном порядке *включают эти работы в плановую деятельность* департаментов и отделов, по руководителям проектов и непосредственным исполнителям.

В заключение можем с уверенностью сказать, что эффективное применение современных стандартов ISO 9000:2000, ISO 12207, ISO 15504, СММ, CMMI и других, о которых говорилось выше, позволяет поставить разработку ПО на промышленную основу, повысить управляемость ключевых процессов и производственную культуру в целом, гарантировать качественную работу и исполнение проектов точно в срок.

13.10. Лабораторная работа № 11 «БПЛА: приложение для видеонаблюдения»

13.10.1. Цель лабораторной работы

Демонстрация процесса разработки практического приложения.

13.10.2. 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.

Все эти три подзадачи были разобраны в лабораторных работах № 9 и 10. Теперь надо объединить подзадачи в итоговое приложение. Мы не будем включать в итоговое приложение подключение к Интернет по модему, при желании это легко сделать по аналогии. У нас опять есть несколько вариантов:

1. Программа на скриптовом языке, например Perl или даже bash. Это наиболее быстрое и простое решение, но с плохими перспективами в плане развития.
2. Обычная linux-программа на C – компромисс между предыдущим и следующим вариантами.
3. QT-приложение с графическим интерфейсом даст использовать потенциал MeeGo на полную мощность.

Выбираем вариант 3. Проект состоит из трех файлов – camserver.pro, camserver.h и camserver.cpp. Мы унаследуем класс CamServerWidget от QWidget и добавим в него два слота – `grab()` и `start()` и таймер `grabTimer`. Пользовательский интерфейс будет состоять всего из трех кнопок *start*, *stop* и *quit* и мы добавим их прямо в исходном коде, в конструкторе класса, без применения специальных инструментов. Там же свяжем слоты и сигналы: сигнал `clicked()` кнопки quit свяжем со слотом `quit()` всего приложения, чтобы завершать его работу. Сигнал кнопки stop свяжем со слотом `stop()` таймера, чтобы останавливать работу таймера. Сигнал кнопки start свяжем со слотом `start()` нашего класса CamServerWidget. Сигнал `timeout()` таймера свяжем со слотом `grab()` нашего класса, в нем будет вся функциональность.

Слот `start()` у нашего класса нужен только для того, чтобы первоначальный запуск таймера происходил сразу же, через 0 миллисекунд, а не через 6000 миллисекунд (6 секунд), которые задаются для каждого последующего запуска таймера. При этом установку нулевого таймаута нельзя перенести из слота в конструктор, потому что в этому случае нулевой таймаут сработает только при первом нажатии кнопки *start*, а если мы остановим процесс и запустим его заново, то таймаут уже окажется ненулевой.

Слот `grab()` вызывается таймером с заданной периодичностью и сначала вызывает приложение для захвата кадра, а затем cUrl для отправки файла на сервер. Для обеих операций установлены таймауты, чтобы суммарное время выполнения не превышало интервал таймера.

Собирать это приложение мы будем так же прямо в MeeGo. Установим зависимости

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

Собираем с помощью

```
qmake; make
```

Перед запуском убеждаемся, что приложение v4l2grab находится в той же директории, камера подключена, веб-сервер доступен по сети. Запускаем приложение и нажимаем кнопку *Start*. В браузере проверяем, что картинка на нашем сервере обновляется каждые 6 секунд. БПЛА можно отправлять на задание.

Конечно, созданное приложение является всего лишь прототипом. Однако и с его помощью можно сделать многое – уточнить требование к приложению, проверить совместимость аппаратуры, оценить требуемые ресурсы, быстро получить тестовое окружение, провести убедительную демонстрацию. Наконец, прототип можно усовершенствовать и постепенно превратить в полноценное приложение.

13.10.3. Задания для самостоятельной работы

1. Добавить в Qt-приложение окно с журналом событий и текущим кадром
2. Создать консольную версию Qt-приложения, способную работать без XWindows и управляемую из командной строки

13.11. Выводы

В лекции были всесторонне рассмотрены вопросы качества разработки программного обеспечения. В лабораторной работе закончено разработка прототипа системы по захвату, сохранению и передаче виде с камеры БПЛА в ЦОД.

13.12. Контрольные вопросы

1) Каким образом заказчик может оценить степень готовности предприятия работать в соответствии с «принципами качества»? Какими стандартами он может воспользоваться для этого?
2) Что включает в себя понятие «качество программного продукта»? Из каких составляющих оно состоит?
3) В чём заключается основная причина появления некачественного программного обеспечения?
4) Какие базовые принципы положены в основу семейства стандартов ISO 9000 модели 2000 года?
5) Для каких целей предназначен стандарт ISO/IEC 9126:1993 и из каких частей он состоит?
6) Что такое стандарт TicIT и с какой целью он был разработан?
7) Какой базовый принцип положен в основу стандарта ISO 12207?
8) Как можно реально управлять процессом разработки ПО?
9) Что такое метрика? Какие классы метрик существуют?
10) Чем можно измерить сложность программного продукта?
11) Как расшифровывается аббревиатура CMM и какие основные принципы положены в основу стандарта?
12) Что в целом должна иметь компания, разрабатывающая программное обеспечение, для получения сертификата 3-го уровня CMM?
13) Что такое программометрика, и какие задачи она решает?
14) На базе каких международных стандартов производится *аудит, совершенство-вание и оценка зрелости процесса разработки ПО* (не зрелости компании, а зрелость процессов)?
15) Нужно ли учитывать расходы на *«качество»* в совокупной цене программного продукта?

16) На каком уровне управления компанией происходит выработка управленческих решений в области менеджмента качества ПО и почему?
17) С какой целью формируется профиль программной или информационной системы?
18) Чем модель CMM отличается от модели CMMI?
19) Чем отличается непрерывное представление ключевых процессных областей в модели CMMI от поэтапного?
20) Каков наиболее рациональный путь формирования стандартного процесса разработки программного обеспечения в масштабах компании?

Список литературы

По лекции № 13

1. Бобровский С. Программная инженерия. – СПб:., Питер, 2003, 222 с.
2. Терехов А.Н. Технология программирования. М:, Интернет-Университет Информа-ционных Технологий. БИНОМ, 2006, 148 с.
3. Wheeler Sh., Duggins Sh. Improving software quality – ACM Proceedings of the 36th annual conference on Southeast regional conference, April, 1998.
4. Kiyaev V.I., Terekhov A.A.. Software Process Improvement in Russian Company: a Case Study // Proc. of the International Workshop New Models of Business. Managerial Aspects and Enabling Technology". – St. Petersburg. 2001. – p. 122-130.
5. Фатрелл Р., Шафер Д., Шафер Л. Управление программными проектами: достижение оптимального качества при минимуме затрат – М., «Вильямс», 2003, 986с.
6. ГОСТ Р 28195-89. Оценка качества программных средств. Общие положения.
7. ISO/IEC 9126:1991, Information Technology – Software Product Quality Characteristics.

8. Боэм Б. и др. Характеристики качества программного обеспечения. – М.: Мир, 1981.
9. Липаев В.В. Качество программного обеспечения. М. Финансы и статистика, 1983.
10. Технологии оценки качества программных продуктов (http://www.rol.ru/news/it/ press/ cwm/25_96/teh.htm)/
11. Sedigh-Ali S., Ghafoor A., Paul R. A. Metrics – Guided Quality Management for Compo-nent-Based Software Systems - Proceedings of the 25th Annual International Computer Software and Applications Conference (COMPSAC).
12. Международный стандарт ISO 9000-1-94. Часть 3: Руководящие указания по примене-нию ISO 9001 при разработке, поставке и обслуживании программного обеспечения. – М., 1996, ИПК, изд-во стандартов. – 34 с.
13. ISO 8402:1994 Quality management and quality assurance – Vocabulary, 1994.
14. Information Technology. Software Life Cycle Processes. ISO/IEC 12207,1995.
15. Зелинский С. IT-Менеджер, №3, 2004.
16. Harter D. E., Slaughter S. A. PROCESS MATURITY AND SOFTWARE QUALITY: A FIELD STUDY - Proceedings of the twenty first international conference on Information systems December 2000.
17. Slaughter S. A., Harter D. E., Krishnan M. S. Evaluating the Cost of Software Quality, - Communications of the ACM, August 1998/Vol. 41, No. 8.
18. Herbsleb J., Carleton A., Rozum J., Siegel J., Zubrow D. Benefits of CMM-Based Software Process Improvement: Initial Results. - Technical Report CMU/SEI-94-TR-013.
19. Furlonger J. ISO 9000 Is No Guarantee of Quality: Research Note Tactical Guidelines.- Gartner Group, August 2000.
20. M.C.Paulk, C.V.Weber, B. Curtis, M. B. Chrissis et al. The Capability Maturity Model: Guidelines for Improving the Software Process. – Addison-Wesley, 1995. – 442 p.

21. Терехов А.А., Туньон В. Современные модели качества программного обеспечения. – BYTE (RE), 1999, № 12. – с. 30-35.
22. CMU/SEI-93-TR-024 "Capability Maturity Model for Software, Version 1.1 – www.sei.cmu.edu/ publications/documents/93.reports/ 93.tr.024.html
23. CMU/SEI-93-TR-025 "Key Practices of the Capability Maturity Model, Version 1.1 – www.sei.cmu.edu/ publications/documents/93.reports/ 93.tr.025.html
24. Кияев В.И. О терминологии и требованиях международного стандарта качества разработки программного обеспечения // В сб. «Системное программирование» (под ред. проф. А.Н.Терехова). Изд. С.-Петерб. ун-та, 2004, с. 311-334.
25. Software Process Improvement Capabilities and determination (SPICE). ISO/IEC TR 15504-CMM, 1998.
26. Paulk M.C. How ISO 9001 Compares with the CMM. IEEE Software, January 1995. – p.74-83.
27. Козодаев А.А. Введение в CMMI (http://www.interface.ru/misc/cmmi.htm)
28. Денис М. Ахен, Арон Клауз, Ричард Тернер. CMMI: Комплексный подход к совершенствованию процессов. – М: МФК, 2005, 330 с.

По лабораторной работе № 11

29. Проект Video4Linux. http://linux.bytesex.org/v4l2/
30. Проект gStreamer. http://www.gstreamer.net/
31. Проект cUrl и libcUrl. http://curl.haxx.se/
32. Проект Apache http://www.apache.org/
33. Проект PHP http://www.php.net/
34. Проект JPEG library http://www.ijg.org/
35. Введение в PPPD. http://en.wikipedia.org/wiki/Point-to-Point_Protocol_daemon

14. Коммерциализация программных приложений

Коммерциализация, инновации, предпринимательство. Пути коммерциализации мобильных приложений. Подготовка и реализация эффективного StartUp-а. Intell AppUp.

14.1. Коммерциализация, инновации, предпринимательство

Коммерциализацию программных приложений для мобильных устройств следует рассматривать в более широком аспекте, чем просто разработку и продажу несложного программного продукта. Не всякий новый продукт содержит в себе явную потребительскую ценность, и не всегда действительно хорошее приложение может быстро найти своего потребителя. Понятие «коммерциализация» тесно связано с понятиями «инновации», «предпринимательство» и «качество» (Рис. 14.1.1).

Рис. 14.1.1. Взаимосвязь между понятиями «инновации», «предпринимательство», коммерциализация» и «качество».

Действительно, программный продукт не найдет дорогу на рынок, если он не будет обладать новой ценностью (New Value) для пользователя, если не будет благоприятной возможности (Facility) для его реализации именно в этот период времени и если он не будет содержать набора функциональностей, которые в достаточной мере удовлетворят ожиданиям потребителя (Consumer Expectations). Таким образом, коммерциализация продукта как создание новой ценности и подчинение деятельности целям извлечения прибыли является сложным и многофакторным процессом, тесно связанным с процессами воплощения идеи в программном продукте (наличие идеи, понимание возможности, поиск единомышленников, создание команды, привлечение финансирования, реализация продукта), защита интеллектуальной собственности, вывод продукта на рынок, оценка его перспектив. А это и есть то, что составляет базовую основу предпринимательства.

Здесь уместно задать несколько вопросов. Предпринимательство и бизнес – это одно и то же? Может ли любой человек стать предпринимателем? Предпринимателями рождаются или становятся? Можно ли этому научиться? И что для этого нужно делать? Вопросы, действительно, важные, попробуем ответить на них в общих чертах.

Очевидно, что с развитием экономики образ человека, открывающего и развивающего свой бизнес, претерпевал серьезные изменения. В Табл. 14.1 приведены сравнительные характеристики бизнесмена 90-х годов и предпринимателя середины 2000-х [1]. Разница налицо! Не последнюю роль сыграли в этом информационные технологии, привнесшие в бизнес культуру открытости [2].

Таблица 14.1. Характеристики делового человека 90-х годов XX века и современного предпринимателя

90-е годы	Настоящее время
Учредитель предприятия малого бизнеса, являющийся, как правило, его собственником	Полноценный предприниматель, организующий эффективный Start Up или имеющий эффективную компанию
Единоличный босс	Признанный лидер
Действует в одиночку или с	Действует в сообществе

небольшой группой	единомышленников и партнеров
Замкнутый, скрытный, идет на контакты в	Открытый, любознательный, пытливый
Полагается, в основном, на свои силы	«NetWorker» (человек, формирующий деловые контакты путем неформального социального общения)
Использует ситуацию, когда видит, что она складывается благоприятно для него	Ищет неформальные возможности, видит их и использует для создания новой ценности
Действует по наитию (интуитивно)	Действует согласно бизнес плану
Собственные, часто моментальные решения	Согласованные решения
Собственность преимущественно у мужчин	Смешанная совместная собственность (учредителями в разных долях могут быть и мужчины, и женщин)

Раньше деятельность любой компании, относящейся к категории предприятия малого бизнеса, автоматически относили к предпринимательской, а человека, открывшего компанию, считали предпринимателем. Однако к концу XX столетия пришли к пониманию, что эта деятельность будет предпринимательской, если в результате появляется продукт, содержащий новое решение (Innovation) и новую ценность (New Value), созданная на основе инновации. К тому же такая деятельность служит для создания новых ниш рынка и формирует «своего» потребителя.

Ярким примером предпринимательской деятельности в 70-е годы могут послужить системы быстрого питания (Fast Food), которые быстро завоевали мир. Они отличались от небольших классических кафе (предприятий малого бизнеса, чаще всего на семейном подряде) тем, что предложили новую потребительскую ценность − экономию времени потребителя и повышение его настроения − основанную на высокотехнологичном оборудовании для быстрого изготовления стандартных блюд, хорошем сервисе, доброжелательном персонале. Такая модель существенно повысила эффективность бизнеса, сформировала новый рынок и нового потребителя!

Ввиду того, что с развитием экономики и технологий, появлением новых бизнес-моделей точка зрения (фокус) на

предпринимательство постоянно изменялась, сложилось несколько определений понятия предпринимательства (Табл. 14.2).

Таблица 14.2. Определение понятия «предпринимательство».

Определение	Источник
Получение прибыли, действуя в условиях неопределенности и риска	F. Knight (1921)
Реализация новых комбинаций в организации фирмы − новых продуктов, новых услуг, новых источников сырья, новых методов производства, новых рынков, новых форм организации	J. Schumpeter (1934)
Действие в условиях неопределенности, координирование производственных ресурсов, введение инноваций и предоставление капитала	B. Hoselitz(1952)
Целенаправленная деятельность с целью инициации и развития бизнеса, ориентированного на извлечение прибылей	A. Cole (1959)
Умеренное принятие риска	D. McClelland(1961)
Решения и оценки в управлении ограниченными ресурсами	M. Casson(1982)
Создание новых организаций	N. Gartner (1985)
Преследование благоприятной рыночной возможности безотносительно доступных в настоящий момент ресурсов	H.Stevenson, M.Roberts, & H.Grousbeck(1989)
Преследование благоприятной рыночной возможности безотносительно доступных в настоящий момент ресурсов, но с учетом предшествующих результатов выбора, сделанных основателями, и их опыта в индустрии	M.Hart, H.Stevenson, & J.Dial (1995)

Приведенные в таблице определения не дают, конечно, полного определения понятия предпринимательства, так как они ограничены временны*ы*ми, экономическими и технологическими рамками. Однако видно, что более поздние определения включают ключевые слова «благоприятная рыночная возможность». В этом смысле по нашему мнению в настоящее время наиболее адекватное определение содержит книга Питера Дракера «Innovation and Entreprenership» (Harpers Collins, 1985): «*Предпринимательский менеджмент − удовлетворение новых потребностей и желаний пользователей, решение новых проблем потребителей посредством*

использования новых возможностей». Об этом позже говорили Джеффри Тиммонс и Стефан Спинелли в книге «New Venture Creation: Entrepreneurship for the 21-st Century» (2003 г.): «*Предпринимательство – это образ мыслей, рассуждений и действий, всецело захваченный открывающейся возможностью реализации, стремящийся к целостному подходу и сбалансированному лидерству*».

Используя определение A. Cole (1959), можем сделать вывод, что конечной целью цивилизованного предпринимательства является извлечение прибыли методами и средствами, не противоречащими нравственным и юридическим законам, социальной основе бытия и развития общества.

Проанализировав приведенные выше определения можно сформулировать наше определение предпринимательства следующим образом: «*Предпринимательство – это процесс создания <стоимости / ценности> (Value Creation) благодаря нетрадиционному образу мышления, набору ресурсов, позволяющих воспользоваться благоприятной возможностью»*. Таким образом, предпринимательство является цивилизованным способом отъема денег у потребителя путем вывода на рынок качественного уникального продукта, обладающего новой потребительской ценностью.

Это определение содержит ключевые фразы, определяющие реалии нового времени: «создание новой потребительской ценности», «качественный уникальный продукт», «вывод продукта на рынок (коммерциализация)», «цивилизованный способ извлечения прибыли». Отсюда следует, что предприниматель – это человек, видящий, распознающий и использующий благоприятные рыночные возможности, реализация которых требует ресурсов больше, чем он располагает в настоящий момент, для создания новой (инновационной) ценности. При этом совокупность ресурсов ничем не ограничивается заранее – это могут быть ресурсы материальные, финансовые, информационные, компетентностные, интеллектуальные, правовые и т.д. Отсюда следует также, что не всякий человек, занимающийся бизнесом, является предпринимателем.

Из определения П. Дракера видно, что предпринимательству можно обучать, т.е. вырабатывать необходимые компетенции для реализации предпринимательского менеджмента.

Выше было показано, что предпринимательство имеет своей частью инновационную составляющую. Что же такое инновация? В книге «Проектирование бизнеса. Почему проектное мышление есть будущее конкурентное преимущество» Роджер Мартин (R. Martin «The Design of Business. Why Design Thinking is the Next Competitive Advantage, 2009) дал очень ёмкое определение понятию «инновация»: «Говоря об инновации, мы говорим о том, чтобы видеть мир не таким, каков он есть, а таким, каким он мог бы быть». В общем случае, инновация есть полученное превосходство на рынке как результат процесса, состоящего из следующих стадий: инвестирование ресурсов в разработку нового знания (научное исследование, проектная разработка);

- получение и осмысление нового знания, реализация технологии или решения на его основе;
- внедрение и/или коммерциализация нового знания (инновационной разработки) в процессы жизнедеятельности человека;
- получение новой ценности или преимущества перед аналогами (если они есть) после внедрения нового знания
- отсутствие в практике или на рынке таких же эффективных аналогов в период до 3-х лет.

Инновационный продукт перестает быть инновационным при наполнении рынка продуктами с аналогичными свойствами и характеристиками. Отсюда возникает вопрос: где и как предприниматель отыскивает и использует инновации? Можно указать на шесть источников инновационных возможностей [3]:

- *непредвиденное* – неожиданное озарение идеей, незапланированный успех или неудача, нестандартное событие;
- *несоответствие* – различие между тем, «что есть» и тем, как это представляется, как это «должно быть»;
- *насущная потребность* – необходимость, связанная с работой (поставленной задачей), которую нужно выполнить для получения устойчивого конкурентного преимущества;

- *изменения в структуре рынка* – явления, связанные с быстрым ростом, конвергенцией возможностей и трансфером технологий, появлением новых бизнес-моделей;
- *демография* – изменения в численности, возрасте, составе, уровне образования и доходов населения существенно влияют на то, *что, когда, где и в каком объёме* будут покупать;
- *изменения настроений и восприятия* – формирование и развитие культуры общества, вкусов, моды, привычек;
- *новые знания* – сфера отражения и закрепления результатов развития науки, техники, методологий и технологий, на базе которых создаются новые понятия, идеи, новаторские предложения и разработки.

Инновационный процесс – упорядоченная, планируемая и регулируемая деятельность по обеспечению разработки новых продуктов (инноваций), их коммерциализации. Это сложная динамическая последовательность действий, связанных с обеспечением зарождения, преобразования и использования инноваций для создания новых потребительских качеств и благ, получения прибыли, достижения конкурентоспособности. Инновационный процесс включает:

- формирование инновационной идеи;
- поиск инвестиций в научно-исследовательские и опытно-конструкторские разработки (НИОКР), в создание нового знания, интеллектуального продукта, в бизнес-модель, техническую (технологическую) разработку, изделие, изобретение.
- инициация инновации, создание условий для реализации инновационной идеи;
- процесс научного исследования и создания собственно интеллектуального продукта;
- защита интеллектуальной собственности;
- отбор и аудит на возможность коммерциализации наиболее перспективных разработок;
- маркетинг инновационной продукции и оценка экономической эффективности;
- Start-Up – организация и становление будущего производства;
- выпуск (производство) инновации;

- коммерциализация (бизнес-реализация) инновации, вывод продукта на рынок;
- продвижение инновации, информирование и рекламирование инновационного продукта;
- диверсификация, более широкое распространение инновации.

Жизненный цикл инновации представляет собой совокупность взаимосвязанных процессов, образующих замкнутый, завершенный оборот развития в течение какого-либо промежутка времени и имеет следующие стадии:

- «долина смерти»: этапы инициации создания инновации и проведение НИОКР. Затраты не дают быстрой отдачи. Используется рисковое инвестирование, частый случай − инвестирование венчурным капиталом. Риск заключается в значимой вероятности получения в будущем не коммерциализуемой разработки;
- вывод на рынок продукта (инновации) и начального получения прибыли после компенсации затрат на инициацию и НИОКР;
- бурный рост продаж нового изделия (продукта, услуги), становление и развитие рынка;
- достижение стабильного состояния максимальных продаж и/или получения максимальной прибыли от производства и реализации продукта. Часто завершается стагнацией;
- появление на рынке аналогичных продуктов. Стадия падения продаж. В некоторых благоприятных ситуациях возможно повышение объема продаж за счет усилий по пролонгации (агрессивная реклама, информирование о новых свойствах этого же товара, маркетинговые мероприятия по подогреву спроса и т. д);
- выхода товара с рынка, поиск новых идей и возможностей для формирования инновации.

В некоторых случаях жизненный цикл инновации для предпринимателя может быть завершен сразу после стадии НИОКР − это, как правило, коммерциализация полученного продукта посредством трансфера технологии, полной или частичной передачи прав на продукт, продажа авторских прав.

Следует отдавать себе отчёт, что наукоемкий высокотехнологический продукт не может быть инновационным

длительное время. Инновационным он может быть весьма непродолжительное время, например, во время стадии выведения на рынок и начала роста объема продаж − до ситуации наполнения рынка аналогом, который производит конкуренты. Как показывает опыт, срок внедрённой инновации составляет от одного до трёх лет – именно такой срок нужен конкурентам для массового создания аналогов инновационного продукта (китайские производители достаточно быстро «подхватывают» производство инновационной продукции и быстро выводят на рынок множество аналогов, включая прямые подделки). Тем не менее, по завершении периода, когда внедрённая разработка перестала быть инновацией, она может реально сохранять статус наукоёмкой и высокотехнологичной, на базе которой могут создаваться и развиваться новые знания и инновационные идеи.

Отметим, что создание инноваций и новых деловых возможностей, основанных на новых знаниях, является наиболее сложным процессом, который сопровождается наибольшей неопределенностью и, соответственно, наибольшими рисками [4]. Именно в этом случае для инновационного предпринимателя чрезвычайно важно иметь соответствующий уровень и набор компетенций для формирования и реализации предпринимательского менеджмента.

14.2. Пути коммерциализации мобильных приложений

Деловая среда нашего столетия существенно отличается от среды последней четверти XX века. Современному бизнесу присущи следующие важнейшие характеристики:

- глобализация знаний и технологий;
- увеличивающаяся скорость изменений в способах ведения бизнеса и технологиях реализации;
- растущие запросы потребителей;
- быстрое изменение требований, правил, законов;
- новые бизнес-модели;
- рост разнообразия и сложности продуктов и сервисов;

- возрастающая конкуренция;
- всё укорачивающийся жизненный цикл продукта.

Всё это особенно характерно для индустрии разработки программного обеспечения. Компании, создающие сложное ПО, идут навстречу требованиям бизнеса, который всё в большей и в большей степени использует информационные технологии и реализующие их программные продукты, выпускают линейки продуктов, причём период времени между появлениями новых версий постоянно сокращается. Быстро изменяются концепции разработки и использования программного обеспечения − это адаптивность и масштабируемость, модульность и платформонезависимость, декомпозиция бизнес-процессов, распределенность информационных систем и централизация ресурсов, интеграция с внешними автоматизированными системами сбора и обработки информации и широкое использование Web-сервисов и систем связи.

Другой отличительной особенностью первого десятилетия XXI века является появление программных платформ для разработки пользовательских приложений для мобильных устройств − телефонов, смартфонов, нетбуков, различного вида сэт-боксов для управления бытовой техникой.

Есть три пути коммерциализации идеи – работать в одиночку, собрать команду и создать компанию (подготовить и запустить Start Up).

Работа в одиночку.

Работа в одиночку или в небольшой группе подразумевает индивидуальную работу каждого участника проекта, приложение разрабатывается практически без предварительного плана, распределяется работа, но не роли. Приложение собирют, тестируют в контекстном режиме и объявлют готовым к использованию. Качество такой работы определется исключительно квалификацией разработчика (разработчиков) и сложностью приложения. Для реализации качества здесь можно применять методологию непрерывного отслеживания правильности разработки (All Steps Tracking and Oversight) или парную работу, часто применяюмую в методологии XP (eXtreme Progamming).

Готовое пользовательское приложение можно выставить в Интернет-магазины Apple App Store или Intel®AppUp Center для последующей продажи.

Apple App Store является одним из наиболее популярных Интернет-магазинов пользовательского программного обеспечения. Apple App Store реализует модель продажи программ, успешно применяемых для мобильных устройств Apple, настольных компьютеров, ноутбуков и планшетников.

Рис. 14.2.1. Инструкция для подключения iPhone к Apple App Store.

Пользователи могут предоставлять или скачивать бесплатные и платные приложения, которые разбиты на категории (обучение, игры, графика, утилиты). Страница каждой программы имеет описание, скриншоты и раздел с пользовательскими отзывами. Для работы с Apple App Store используется iOS SDK + Mac OS XSL. У пользователей Mac OS X 10.6 Snow Leopard программа для работы с магазином появится после установки очередного обновления системы. В следующую версию Mac OS X 10.7 Lion, выход которой запланирован на лето 2011 года, поддержка сервиса будет встроена

изначально. Для осуществления покупок применяется учетная запись Apple ID, которая также используется для доступа к iTunes Store. Пользоватья доступом к ресурсам магазина чрезвычайно просто – на рисунке 14.2.1 показана пошаговая инструкция для подключения iPhone к Apple App Store.

В Apple App Store действует схема поощрения разработчиков – автор размещенной в интернет-магазине программы получает 70% от продаж (оставшиеся 30% составляет комиссия для компании Apple). С момента открытия в 2008 году число скачиваний к концу 2010 года достигло 10 миллиардов. Компания Apple зарегистрировала сочетание слов APP Store в качестве товарной марки.

Такой успех, понятно, не остался без внимания! Компания Microsoft подала в Офис Патентов и Торговых Марок США (USPTO) заявление с требованием отобрать у компании Apple право эксклюзивного использования сочетания App Store. Наименование App Store было зарегистрировано Apple в качестве торговой марки для Интернет-магазина приложений для мобильных устройств на основе iOS сразу после запуска сервиса в 2008 году. Однако, Microsoft считает, что такое название не может использоваться одним владельцем, поскольку является общеупотребительным термином. App Store обозначает, согласно заялению Microsoft, просто Интернет-магазин мобильных приложений. Таким образом, название App Store нельзя регистрировать в качестве торговой марки, поскольку из-за этого другие компании не смогут использовать это сочетание даже просто для описания своих предложений.

Не менее популярным в настоящее время является аналогичный нитернет-магазин компании Intel – Intel App Up Center (видим, что здесь Intel изящно решил проблему с использованием слов «App Store»). Intel®AppUp Center − это новый сервис, предоставляющий пользователям каталог приложений для нетбуков и персональных компьютеров с возможностью покупки и загрузки (www.appup.com/applications/index). AppUp центром поддерживаются платформы Windows XP & Windows 7, Moblin™, MeeGo 1.1. На Рис. 14.2.2. показана страница магазина, а на Рис. 14.2.3 схема взаимодействия с Intel App Up Center.

Рис. 14.2.2. Интерфейс Интернет-магазина Intel App Up Center.

Рис. 14.2.3. Схема взаимодействия с Intel AppUa Center.

Алгоритм доступа в магазин несложен и состоит из нескольких простых шагов:

Рис. 14.2.4. Страница для скачивания программы Intel® Atom™ Developer Program.

1. Прочитайте описания программы Intel® Atom™ Developer Program.
2. Выберете ОС Windows / (Moblin→MeeGo), средства разработки (см. Лаб. Работу по установке MeeGo на нетбук).
3. Получите логин в Программе.
4. Скачайте SDK.
5. Придумайте уникальное имя приложению и получите GUID на него.
6. Разработайте и протестируйте приложение.
7. Зарегистрируйтесь в программе (начальные установки практически не требует подробных данных о вас).
8. Отправьте Приложение.
9. Следите за статусом валидации Приложения.

- Intel проверяет только корректность работы с интерфейсами и выполнение общих требований к приложению
- общее и контекстное тестирование приложения за Вами

В Intel AppUp предполагается, что у одного пользователя может быть до пяти разных устройств под одной лицензией. Внутри системы AppUp есть процесс валидации − когда программа загружается в магазин на продажу, разработчик указывает, что его программа, например, предназначена для мобильных телефонов и нетбуков. В центре валидации приложений проверяется, действительно ли приложение работает на заявленных типах устройств, и если выясняется, что, например, разработчик заявил поддержку ТВ, а на деле программа только запускается на ТВ, но органы управления телевизором не работают и управлять ей невозможно, то программа автоматически исключается из этой категории. Будет ли это одно приложение для всех сегментов или разные модификации приложения для различных устройств, зависит от разработчика. Разработчик сам следит за статусом продаж, оценками приложений и динамикой своего общего рейтинга. Это поможет емуу правильно оценить востребуемость приложения и его пользовательскую ценность.

Собрать команду.

В этом случае коммерциализовать перспективную идею можно как в предыдущем случае, предлагая разработанное приложение в какой-либо Интернет-магазин, либо работать под заказ. Вторая ситуация не такая простая, как разработка «фантазийного» приложения для пользователя вообще − «кому понравится, тот заплатит и скачает». В этом случае требуется полноценная проектная работа со всеми вытекающими «проектными» требованиями. И самое первое, с чем сталкивается разработчик идеи – это формирование адекватной проектной команды.

Проектная команда – это совокупность единомышленников, связанных общей работой или деятельностью, формируемая для достижения определенной цели, которая на время выполнения проекта становится общей целью для команды. И один из самых сложных вопросов для первичного становления команды – это формирование единства. Есть наборы общих правил, которые могут помочь инициатору идеи преодолеть этот сложный этап [7].

1. *Шаги по построению команды:*

- примите решение сформировать команду, соберите лучших игроков
- вооружите членов команды ответственностью и полномочиями – это выделяет и подготавливает лидеров
- доведите идею до всех членов команды, убедитесь, что каждый воспринял её
- обсудите действия по реализации идеи, нацельте всех членов команды на успех
- старайтесь работать вместе – это обеспечит общность команды
- оценивайте общий успех команды – это повышает её моральное состояние
- следите за тем, чтобы вклад каждого в деятельность команды окупался
- прекратите вклад в развитие тех, кто не хочет расти
- создайте новые возможности для команды, планируйте её развитие
- предоставьте команде наилучший шанс для успеха

2. *Единство команды:*

- формулирование ценностей, общих для всех членов команды
- понимание миссии организации и видение задач
- настройка личных качеств и действий на достижение общей цели
- развитие навыков коммуникации
- правильная оценка способностей каждого и рациональное разделение ролей
- уверенность в «добрых намерениях» других членов команды
- уважение различных мнений
- политика открытых дверей

3. *Правила работы в команде:*

- верь, что можешь изменить мир
- твори и создавай
- оценивай, когда работать самому и когда работать в команде
- доверяй своим коллегам, делись идеями
- умей оценивать риск и обоснованно рисковать

- работай быстро и качественно, никогда не «запирай» инструменты
- верь, что вместе мы можем сделать все
- нет такого понятия «Этого нельзя сделать!»

4. Модели поведения, которые способствуют успеху команды:

- умеют создать отношения взаимного доверия, уверенности и приверженности делу среди членов команды
- готовы обсуждать всей группой цели, планы и график работы, оставляя место для разногласий и поиска консенсуса
- каждыйобщается с каждым и каждый выполняет свою работу
- поощряют конструктивную критику и взаимопомощь
- поддерживают и уважают своих коллег, не предъявляя к ним слишком больших требований
- понимают, что ответсвенность команды – это ответственность каждогоеё члена.

Подчеркнём, что между группой, работающей сообща, и командой есть существенные различия (Табл. 14.1):

Таблица 14.2.1.

Группа	**Команда**
Члены группы работают самостоятельно, не интересуясь общей целью	Члены команды понимают, что целей лучше всего достичь общими усилиями
Индивиды без нужды обращают внимание на себя	Члены команды испытывают чувство принадлежности к общей работе и команде. Они сами участвовали в определении целей
Члены группы получают приказы без учета их мнения	Члены команды осуществляют свой вклад в успех организации, так как их идеи получают должное внимание
Доминирует недоверие к коллегам, чувство апатии и равнодушия	Атмосфера доверия. Существует открытый обмен идеями, мнениями, недовольством, чувствами

Нет взаимопонимания, это ведет к появлению интриг и ведет к образованию внутренних групп, объединенных личными симпатиями или корыстными целями	Открытые и честные отношения, прилагаются усилия понять точку зрения другого
Есть возможность получить хорошую квалификацию, но ее применение ограничивается начальством	У членов команды есть стимул развивать свои умения и прилагать свои знания в работе. Получают поддержку команды
Члены группы попадают в конфликтные ситуации, которые не знают, как разрешить	Конфликты − нормальная часть человеческого общения. Такие ситуации воспринимаются как возможность реализации новых идей
Процесс принятия решений часто осуществляется без участия членов группы	Члены команды принимают участие в процессе принятия решений

Только такая команда, построенная на показанных выше принципах, поможет инициатору идеи добиться своей мечты и достичь желаемых вершин! А как должна работать такая команда, чтобы реализовать качественный продукт? В этом случае рационально применять две широко апробированные и отлично себя зарекомендовавшие методологии MSF (Microsoft Solution Framework) и Scrum.

Методология Microsoft Solution Framework

Корпорация Майкрософт выпустила в свет пакет руководств по эффективному проектированию, разработке, внедрению и сопровождению решений, построенных на основе своих технологий. Эти знания базируются на опыте, полученном Майкрософт при разработке и сопровождению программного обеспечения, опыте консультантов Майкрософт, разрабатывавших проекты на предприятиях заказчиков, и лучшем из того, что накопила на данный момент IT-индустрия. Всё это представлено в виде двух связанных и хорошо дополняющих друг друга областей знаний: Microsoft Solutions Framework (MSF) и Microsoft Operations Framework (MOF) (рис. 14.2.3).

Рис. 14.2.3. Взаимосвязь и области применения методологий MSF и MOF.

Microsoft Solutions Framework − это комплект взаимосвязанных моделей, концепций и руководств по созданию и внедрению распределенных информационных систем уровня предприятия [8]. Он содержит набор интегрированных ресурсов (практические руководства, аудиторные занятия, описания методик и методологий) и принципов, приводящих проектные группы к успеху. MSF не является методологией, а скорее предоставляет гибкие и практические пути применения информационных технологий для решения проблем, обеспечивает структуру, помогающую локализовать проблемы и облегчить принятие эффективных решений.

В основе Microsoft Solutions Framework лежат следующие идеи:

- идентификация целей проекта и планирование их достижения;
- формирование факторов успеха;
- управление рисками;
- выпуск промежуточных версий;
- планирование активности каждого члена проектной;
- четко обозначенные контрольные точки (вехи);
- проектные команды небольшой численности.

Подробную информацию по MSF в виде рекомендаций, описаний действий, инструкций, шаблонов документов можно найти на сайте www.microsoft.com/msf/.

Рис. 14.2.4. Области ключевых компетенций MOF.

MOF призван обеспечить организации, создающие критически важные (Mission-Critical) IT-решения на базе продуктов и технологий Microsoft, техническим руководством по достижению их надежности (Reliability), доступности (Availability), удобства сопровождения (Supportability) и управляемости (Manageability). MOF затрагивает вопросы, связанные с организацией обучения и работы персонала, процессов, технологиями и менеджментом в условиях сложных (Complex), распределенных (Distributed) и разнородных (Heterogeneous) IT-сред. MOF основан на лучших

производственных методиках, собранных в библиотеке IT Infrastructure Library (ITIL), составленной Агентством правительства Великобритании (Central Computer and Telecommunications Agency). Информация по MOF доступна в Internet по адресу http://www.microsoft.com/mof/.

Модель процесса в MSF формируется на базе итеративной и эволюционной моделей ЖЦ, основывается на сценариях использования, работы выполняет небольшая команда (хотя есть способы масштабирования команд для больших проектов), используется подход к тестированию, основанный на контексте. Модель полностью ориентирована на заказчика − принцип «качества обслуживания заказчика.

В модели поддерживаются следующие потоки работ (Workflow):

- Формулировка целей и задач проекта
- Идентификация факторов успеха проекта
- Создание сценариев и тестирование сценариев
- Создание требований по качеству обслуживания
- Планирование итераций
- Создание архитектуры решения
- Реализация задачи по разработке
- Сборка продукта
- Быстрое тестирование, исправление и закрытие ошибок
- Тестирования требований по качеству обслуживания
- Выпуск продукта
- Управление проектом

В модели проектной команды MSF у каждого члена команды имеются четко очерченные роли и зоны ответственности (рис. 14.2.5). В основу модели положены следующие принципы:

- взаимозависимые и взаимосвязанные роли в малой команде
- определение роли, особой миссии и зоны ответственности для каждого члена проектной команды
- распределенное управление проектом и ответственность
- каждый сфокусирован на успехе проекта и настроен на работу в течение всего цикла проекта

- эффективные коммуникации между членами команды являются ключевым фактором успеха;
- параллельная работа всех участников команды над проектом;
- пользователи и обучающий персонал включены в команду.

Рис. 14.2.5. Роли членов проектной команды MSF.

В такой группе: общие для всех членов группы цели и конкретные для каждого планы реализации; каждый понимает как проблемы конечного пользователя, так и проблемы разработчика; каждый общается с каждым, и каждый делает реальную работу; каждый несет ответственность за свою работу, в том числе и перед группой.

Менеджер продукта. Обеспечивает коммуникационный канал между заказчиком и проектной командой. Менеджер продукта управляет ожиданиями заказчика, разрабатывает и поддерживает бизнес-контекст проекта. Его работа не связана напрямую с продажей продукта, он сфокусирован на продукте, его задача – определить и обеспечить удовлетворение заказчика. Лучшая кандидатура на эту роль – существующий постоянный пользователь продукта, сотрудник коммерческого отдела или другой представитель заказчика, если он понимает задачи и механику бизнеса.

Менеджер проекта. Управляет коммуникациями и взаимоотношениями в проектной команде, является координатором действий, разрабатывает функциональные спецификации и управляет ими, ведет график проекта и отчитывается по состоянию проекта, инициирует принятие критичных для хода проекта решений.

Архитектор, разработчик. Принимает технические решения, которые могут быть реализованы и использованы, создает продукт, удовлетворяющий спецификациям и ожиданиям заказчика, консультирует другие роли в ходе проекта. Разработчик участвует в обзорах, реализует возможности продукта, участвует в создании функциональных спецификаций, отслеживает и исправляет ошибки за приемлемое время. В контексте конкретного проекта разработчик дополнительно может, например, производить инсталляцию программного обеспечения, настройку продукта или услуги.

Разработка сложных программных систем требует детального знания высокоуровневых языков программирования, визуального моделирования, сетевых технологий и проектирования баз данных. В связи с тем, что один человек не может быть экспертом во всех областях этих технологий, важно, чтобы экспертиза во всех областях реализации проекта была представлена соответствующими техническими специалистами, входящими в группу разработчиков, а руководитель этой группы знал и понимал ключевые моменты каждой из этих технических областей.

Тестер. Тестирование включает в себя не только проверку кода. Тестировать надо функциональные спецификации, систему обеспечения производительности, пользовательские интерфейсы, планы внедрения и используемую терминологию. Тестер обеспечивает возможность того, чтобы все особенности и задачи тестирования были известны до выпуска версии продукта, разрабатывает стратегию тестирования и планы тестирования для каждой из фаз проекта. Планы и процедуры тестирования для сложных программных систем должны быть комплексными.

Инструктор. Отвечает за снижение затрат на дальнейшее сопровождение продукта, обеспечение максимальной эффективности работы пользователя. Важно, что речь идет о производительности пользователя, а не системы. Для обеспечения

оптимальной продуктивности инструктор собирает статистику по производительности пользователей и создает решения для повышения производительности, с использованием таких технологий, как мультимедиа, видео, HTML, встроенные системы подсказки, мастера, тренажеры и т.п. Инструктор принимает участие во всех обсуждениях пользовательского интерфейса и архитектуры продукта.

Логистик. Задача логистика – обеспечить «гладкое» внедрение и развитие продукта. Обычной является ситуация, когда внедрение продукта стоит дороже его разработки. Логистик должен обеспечить такое состояние дел, чтобы заказчик был готов к внедрению, чтобы вовремя были выполнены все подготовительные работы и существовала необходимая инфраструктура.

Помимо перечисленных ролей, можно выделить еще «роли поддержки и сопровождения». Это специалисты и эксперты в ключевых точках инфраструктуры. Они привлекаются к работам, когда это необходимо, но не принимают решений. Если численность проектной комнды меньше шести человек, то часть ролей может совмещаться, и их будет выполнять один человек. MSF дает рекомендации по совместимости нескольких различных ролей. На Рис. 14.2.6 показаны рекомендации по совмещению ролей.

Роли	**Менеджер продукта**	**Менеджер проекта**	**Разработчик**	**Тестер**	**Инструктор**	**Логистик**
Менеджер продукта	⊗	Нет			Да	Да
Менеджер проекта	Нет	⊗	Нет	Нет	Да	Да
Разработчик		Нет	⊗	Нет		
Тестер		Нет	Нет	⊗		
Инструктор	Да	Да			⊗	
Логистик	Да	Да				⊗

Рис. 14.2.6. Рекомендации MSF по совмещению ролей в проектной команде.

В настоящее время существуют наборы программных продуктов для поддержки командной работы. На наш взгляд одним из таких интересных и недорогих наборов является следующий: Visual Studio Team System 2010 (интегрированное средство управления программными проектами), SQL Server 2008 (одно из наиболее эффективных средств для хранения и управления данными) и BizTalk Server 2010 (средство для управления и автоматизации бизнес-процессов), с помощью которого можно полностью реализовать все задачи по разработке мобильного программного приложения небольшой по численности командой.

Методология Scrum

Scrum — это набор принципов, на которых строится процесс разработки, позволяющий в жёстко фиксированные небольшие промежутки времени («спринты») от 2 до 4 недель предоставлять конечному пользователю работающее ПО с новыми возможностями, для которых определён наибольший приоритет. Возможности ПО к реализации в очередном спринте определяются в начале спринта на этапе планирования и не могут изменяться на всём его протяжении. При этом строго-фиксированная небольшая длительность спринта придаёт процессу разработки предсказуемость и гибкость [9].

Методология устанавливает правила управления процессом разработки, применяет итеративную модель и позволяет использовать уже существующие практики кодирования, корректируя требования или внося тактические изменения. Использование этой методологии дает возможность выявлять и устранять отклонения от желаемого результата на более ранних этапах разработки программного продукта, реализуя тем самым заданное качество.

В методологии Scrum имеется всего три роли:

Scrum Master − самая важная роль в методологии. Как правило, эту роль в проекте играет менеджер проекта или team-leader. В обязанности Scrum Master'a входит обеспечение максимальной работоспособности и продуктивности команды, четкого взаимодействия между всеми участниками проекта, своевременное решение всех проблем, тормозящих или останавливающих работу,

ограждение команды от всех воздействий извне во время итерации и обеспечение следования процессу всех участников проекта.

Product Owner − человек, поставляющий требования программистам. Обычно Product Owner является представителем заказчика или представляет рынок, на котором реализуется продукт. Product Owner должен составить бизнес-план и план развития с требованиями. Исходя из имеющейся информации, Product Owner подготавливает список требований, ранжированный по значимости. В обязанности участника проекта входит своевременное предоставление требований к продукту, определение дат и содержания релизов, эффективное управление приоритетами и корректировка требований.

Team − команда, состоящая из пяти–девяти человек. В нее входят люди с различными навыками — разработчики, аналитики, тестировщики. В отличие от методологии MSF здесь нет заранее определенных и поделенных ролей в команде, ограничивающих область действий членов команды. Первая задача такой команды − поставить реально достижимую, прогнозируемую, значимую цель для итерации. Вторая задача − реализовать эту цель в отведенные сроки и с заявленным качеством. Цель итерации считается достигнутой, если все поставленные задачи реализованы, весь код написан по стандартам кодирования, программа протестирована и все найденные дефекты устранены. В обязанности всех членов Scrum-Team входит участие в выборе цели итерации и определение результата работы. Они должны эффективно взаимодействовать со всеми участниками команды, самостоятельно организовывать свою работу, предоставлять владельцу рабочий продукт в конце каждого цикла.

На Рис. 14.2.7 показаны артефакты в методологии Scrum.:

Product Backlog — это имеющихся на данный момент список деловых и технических требований к системе с указанием приоритетов. Product Backlog включает в себя задачи, технологии, проблемы, запросы ошибки и т.д. Элементы этого списка называются «историями» (User Story) или элементами Backlog'а (Backlog Items). Product backlog открыт для редактирования для всех участников Scrum-процесса.

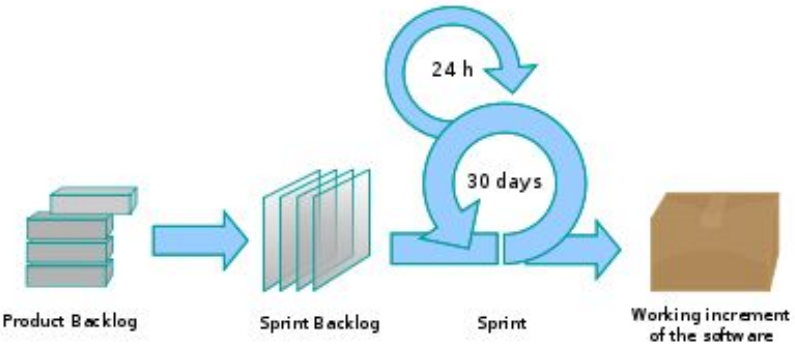


Рис. 14.2.7. Процессы и артефакты вметодологии Scrum.

Sprint Backlog содержит функциональность, выбранную Product Owner из Product Backlog. Все функции разбиты по задачам, каждая из которых оценивается и выполняется командой в соответствии с гибким планом работ.

Burndown chart показывает, сколько уже исполнено и сколько ещё остаётся сделать.

Результатом Sprint является готовый продукт, который можно передавать заказчику. Каждый sprint представляет собой «водопад». В течение спринта делаются все работы по сбору требований, дизайну, кодированию и тестированию продукта.

Такая итеративная разработка, в конце которой создается готовый к использованию продукт, применяется во многих гибких методологиях. Но в Scrum хорошо реализован процесс сбора функций и их распределение по итерациям. Спорный момент здесь – отсутствие жестко заданного распределения ролей и обязанностей в команде. Принцип «все ответственны за всё» работает далеко не всегда, наоборот, четкое распределение ролей позволит сконцентрироваться сотрудникам только на тех работах, где они могут принести максимум пользы.

Как и многие другие гибкие методологии, Scrum требует постоянного участия представителя заказчика или наличия представителя рынка. В обоих случаях всё не так хорошо, как описывается в стартовой модели. Заказчик не всегда может

полностью определить, что именно ему нужно, а частое изменение требований вносит неопределённость и может сильно замедлить работу. Представитель рынка выявляет интересы некоторого количества конкретных покупателей, а определение интересов абстрактной целевой группы возможных пользователей − процесс, требующий продолжительных исследований.

Тем не менее, методология Scrum является одной из самых применяемых гибких (Agile) методологий, когда необходимо в сжатые сроки небольшой командой изготовить и передать заказчику качественный продукт.

И, наконец, третий путь коммерциализации перспективной идеи программного приложения – это организовать компанию «с нуля» и запустить производство продукта на промышленной основе.

14.3. Подготовка и реализация эффективного StartUp-а

Как стать победителем и преуспеть? Рыночная экономика давно знает ответ на этот вопрос – разработчик должен заинтересовать потенциального пользователя в своем продукте. Показать, что он выгодно отличается от других. Для этого продукт должен лучше удовлетворять некие ожидаемые потребности по сравнению с другими. Или открыть потребителю новые потребности, о которых тот, может быть, даже и не догадывался, т. е. создать инновацию. Однако создать инновацию – это полдела! Вторая половина, и не менее важная с точки зрения достижения конечного результата, представить ее на рынке, продвинуть, продать. Для этого необходим начальный толчок, старт, который придаст инновации необходимое ускорение для выхода на рынок. Проще говоря, следует построить Start Up.

Коротко понятие Start Up (стартап) можно определить как процесс создания компании «с нуля». Это стадия развития компании, когда она недавно образована, обладает перспективными идеями, прототипами продукта или опытными образцами, пытается организовать производство и выход продукции на рынок. Применительно к финансированию компании, стартапом могут называть рискованное инвестирование в проект или фирму, либо не

имеющую сколько-нибудь длительной рыночной истории, либо находящуюся на самом раннем этапе продаж.

Таким образом, стартап – это компания, которая родилась из идеи и имеет высокий потенциал превратиться в высокодоходное предприятие. Этим стартап отличается от индивидуальных или групповых проектов малого бизнеса (Табл. 14.3.)

Таблица 14.3. Типология стартапов.

Параметры проекта	Индивидуальный, групповой	Промышленный/ инновационный (высокопотенциальный)
Инициаторы / Основатели	Физическое лицо/Группа лиц	Физическое лицо/Команда
Изначальные инвестиции	Небольшие	По обстоятельствам
Финансовые потребности	Небольшие	Большие
Структура капитала	Закрытая	Открытая
Потенциал роста	Слабый	Высокий
Потенциал экспорта	Слабый	Высокий
Цель	Краткосрочная прибыльность	Устойчивый рост
Степень защиты интеллектуальной собственности	Низкая	Высокая
Зависимость от основателя	Высокая	Довольно слабая
Сфера деятельности	Консалтинг, услуги	Высокотехнологичные товары, программные продукты
Использование научных достижений	Среднее	Существенное

Сложилась классификация стартапов, которая включает несколько ключевых стадий развития стартапа (жизненный цикл стартапа):

Pre Start Up стадия

- подготовительная стадия (pre-seed stage)
- посевная стадия (seed stage)
- идеальный и работающий прототип (prototype & working prototype)
- альфа-версия проекта или продукта (alpha-version)
- закрытая бета-версия проекта или продукта (private beta-version)
- Публичная бета-версия проекта или продукта (public beta-versin)

Запуск проекта в эксплуатацию или продукта в производство

- запуск, или ранняя startup-стадия (launch, or early startup stage)
- стадия Start Up (Star Up stage)
- работа с первыми клиентами, или поздняя Startup-стадия (first clients, or late startup stage)
- вывод продукта на рынок (market stage)

Post Start Up стадия

- стадия роста (growth stage)
- стадия расширения (expansion stage)
- стадия выхода (exit stage)
- Pre-IPO stage (при выходе через проведение IPO — Initial Public Offering, или первичное размещение компанией своих акций на бирже)
- IPO (при выходе на IPO).

Таким образом, стартап имеет совершенно определенный жизненный цикл, который следует планомерно реализовывать. Попытки «перескочить» через отдельные этапы приводят чаще всего к тому, что стартап или вообще не начинается, или рушится на одной из стадий.

Если идти от истоков развития высокотехнологичной мировой экономики, то большинство современных успешных компаний начинались со стартапов. Intel, Apple, Cisko, Sun Microsystems, Google, Yahoo, Facebook, YouTube – 40 лет назад этих компаний не было на бизнес-карте США. А сегодня это гиганты индустрии с миллиардными оборотами!

Как показывает мировой опыт часто идеи для успешных стартапов появлялись случайно. К примеру, компания Lotus начиналась с того, что Митч Капор просто написал удобную компьютерную программу для своего друга, чтобы тот смог привести в порядок свои дела. Стив Возняк, работавший на Hewlett-Packard, хотел создавать компьютеры, но работодатель не давал ему реализовывать свои идеи. Стив ушел из компании и встретил Стива Джобса. Так появилась компания Apple. А Дэвид Фило, коллекционировавший ссылки, даже и не думал, что его увлечение выльется впоследствии в проект «Yahoo».

Самым образцовым и, пожалуй, самым идеальным стартапом конца XX века по праву считается компания Google [6]. Компания стала символом всего лучшего и перспективного, что ассоциируется с знаменитой Кремниевой долиной. Компания была организована Лари Пейджем и Сергеем Брином в 1998 году, и в этот момент в компании было всего 8 сотрудников. За последующие пять лет персонал компании вырос до 8000 человек. В 1999 году выручка составила 200000 долларов. Компания стала прибыльной в 2001 году (прибыль составила 7 миллионов долларов), в 2006 году активы компании оценивались уже в 8 миллиардов долларов. Вот так двум аспирантам Стэнфордского университета удалось за пять с небольшим лет реализовать «американскую мечту» – построить «с нуля» высокотехнологичное и высокодоходное предприятие.

Многие исследователи считают, что стартап – это не просто основание компании и организация выпуска продукта [5-7]. Если исходить из определений понятия «предпринимательство», данных выше, то можно видеть, что организация стартапа чаще всего происходит в условиях большой неопределенности, риска и при наличии ограниченных или даже недостаточных ресурсов. Что же тогда движет организаторами стартапа? Прежде всего – это мотивы самореализации, моменты яркого творчества и получения удовольствия. В ролике, снятом в 2002 году, когда компания Google ещё не получила всемирной известности, Лари Пейдж сказал: «Заработать денег – это, конечно сильный мотив, но для небольшой компании серьёзным фактором часто оказывается стремление и возможность зарабатывать деньги с удовольствием!».

Следующим феноменом известных американских стартапов можно считать количество основателей и их возраст. Чаще всего это были два или три молодых партнера: Уильям Хьюлетт и Дэвид Паккард (HP), Стивен Джобс и Стивен Возняк (Apple), Сандра Лернер и Лен Босак (Cisko), Дэвид Файло и Джерри Янг (Yahoo), Уильям Гейтс и Пол Аллен (Microsoft) и, наконец, троица Роберт Нойс, Гордон Мур и Энди Гроув (Intel). Были и одиночки – Лари Элисон (Oracle), Марк Андриссен (Netscape), Джефф Безос (Amazon), Майкл Делл (Dell), Марк Цукерберг (Facebook) (Табл. 14.4).

Таблица 14. 4. Некоторые данные по выдающимся стартапам и их основателям.

Ком-пания	Год основания	Основатель(и)	Возраст	Откуда родом
HP	1939	Билл Хьюлетт	26	США
HP	1939	Дэвид Пакард	27	США
Intel	1968	Роберт Нойс	41	США
Intel	1968	Гордон Мур	39	США
Intel	1968	Энди Гроув	32	Венгрия
Micro soft	1975	Билл Гейтс	20	США
Micro soft	1975	Пол Аллен	22	США
Apple	1976	Стив Джобс	21	США
Appl e	1976	Стив Возняк	26	США
Oracl e	1977	Ларри Эллисон	33	США
Sun	1982	Винод Хосла	27	Индия
Sun	1982	Билл Джой	28	США
Sun	1982	Энди Бехтольшайм	26	Германия

Sun	1982	Скотт МакНили	28	США
Cisco	1984	Лен Босак	29	США
Cisco	1984	Сандра Лернер	29	США
Netscape	1994	Марк Андриссен	23	США
Amazon	1994	Джефф Безос	30	США
Dell	1984	Майкл Делл	19	США
eBay	1995	Пьер Омидьяр	28	Франция
eBay	1995	Джефф Сколл	30	Канада
Yahoo	1995	Дэвид Файло	29	США
Yahoo	1995	Джерри Янг	27	Тайвань
Google	1998	Ларри Пейдж	25	США
Google	1998	Сергей Брин	25	СССР
Facebook	2003	Марк Цукерберг	19	США

Итак, вы молоды, у вас есть сногсшибательная идея, вера в свои силы и возможности свои в доску партнеры и желание сделать что-то очень полезное для всех людей и для себя лично. Иными словами, вы хотите осчастливить мир! Но с чего начинать и где взять денег на благородное начинание? К кому и как подойти, чтобы возможный инвестор или венчурный капиталист заинтересовались идеей, прониклись её перспективами и дали средства на её реализацию? Ответ достаточно прост – грамотно построить стартап!

Рис. 14.3.1.

Отметим сразу, что в большинстве случаев «грамотный стартап» начинается с формулирования возможностей, стратегии реализации и правильного формирования продуктового предложения, которое «снимает» трещину между желанием инициатором стартапа быстро получить необходимые средства и реальным опасением возможного инвестора потерять эти деньги (Рис. 14.3.1).

Можно привести пример удачного начала, чрезвычайно быстрого взлёта и такого же быстрого понижения. В начале 2010 года Интернет облетела сенсация − новый Интернет- старта ChatRoulette.com (http://venture-biz.ru/informatsionnye-tekhnologii/30-internet-startup-chatroulette), созданный 17-летним московским школьником Андреем Терновским, стал любимой игрушкой миллионов пользователей во всем мире. По оценке Google AdPlanner, к середине марта посещаемость сайта достигла 0,5 млн. человек в день. Пользоваться сайтом было очень легко − нажав на кнопку Start, посетитель видел в одном окне себя, в другом анонимного собеседника, который попадал в это окно случайным образом, а в третьем участники чата могли обмениваться письменными сообщениями.

Успешный стартап привлек бесчисленных бизнес-ангелов − от владельца DST Юрия Мильнера до первого инвестора Twitter Фреда Уилсона. Но А.Терновский отказался продавать даже долю в своей компании. Он нанял семерых программистов и уехал в Калифорнию. Но позже трафик пошел на спад, к концу июня число уникальных пользователей упало ниже 250 000, а концу года интерес к сайту уменьшился ещё больше. Эксперты полагают, что в первые месяцы после старта ChatRoulette.com «снимал сливки», но реальной стратегии у основателя ставшего на короткое время популярным Интернет-ресурса не было.

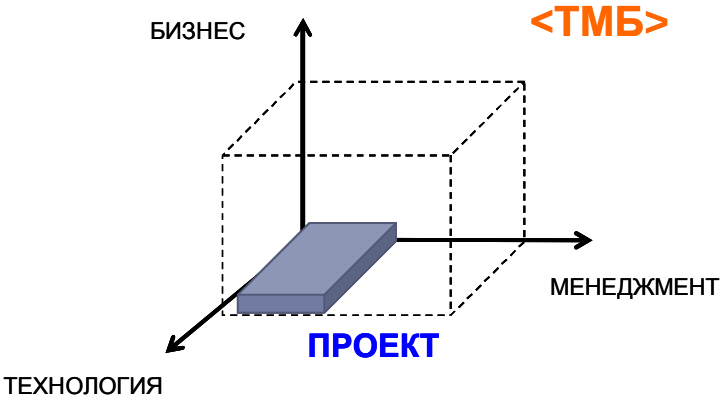


Рис. 14.3.2. Система координат для формулирования продуктового предложения.

Грамотно выбранная стратегия позволяет инициатору стартапа использовать благоприятную рыночную возможность, найти незанятые ниши рынка или создать новые (маргинальные) ниши и наладить производство продукта с новой потребительской ценностью. Грамотно сформулированное продуктовое предложение помогает достаточно быстро найти необходимые средства для изготовления работающего прототипа высокотехнологичного устройства или альфа-версии программного приложения (продукта). Продуктовое предложение следует формулировать в системе координат <ТМБ> с указанием её общих достоинств, ценности продукта для потребителя, имеющихся и возможных рисков, путей технической реализации (Рис. 14.3.2):

Поскольку любая, даже хорошо обоснованная идея проходит экспертный анализ, то возникает вопрос: «Что в первую очередь оценивают эксперты, на что обязательно обращают внимание?». Это следующие ключевые обстоятельства:

- человеческий фактор (качества руководителя проекта, наличие и сыгранность команды);
- правовая ситуация (наличие прав на продвигаемую интеллектуальную собственность, подтверждение прав патентом или свидетельством на изобретение, возможность расширения прав на новые территории и новых собственников);
- важность предлагаемого полезного свойства (ожидания потребителя);
- наличие рынка и точное позиционирование на нем;
- наличие ключевого преимущества (отличающего предлагаемый продукт от аналогичных);
- риски: правовые, научные, технические, страновые, иные;
- наличие и понимание плана отработки и развития проекта (компромисс <возможности-время-ресурсы>);
- наличие и понимание плана входа на рынок;
- понимание потребностей участников проекта и наличие плана их достижения, в частности – выхода из проекта.

Презентация проекта, как правило, начинается с представления команды – следует понимать, что разработчик «продает» проект, а инвестор «приобретает» команду во главе с руководителем.

Отметим типичные ошибки при представлении продуктового предложения:

- отсутствие самого предложения или его подмена совокупностью технических характеристик;
- улучшение второстепенных, не важных для потребителя функций;
- улучшение объектов с низким потенциалом развития;
- невнятное предложение, не выделяющее новые ключевые потребительские свойства (наиболее часто встречаемая «слабость» проекта!)

На Рис. 14.3.3 показаны неправильно (слева) и правильно (справа) представленные продуктовые предложения. В качестве

упражнения предоставляем читателю возможность объяснить причины правильной и неправильной формулировок этого продуктового предложения.

Рис. 14.3.3. Виды продуктового предложения.

Ключевая потребительская ценность – это свойство продукта, из-за которого он востребован. Потребитель приобретает продукт из-за его возможности производить и реализовывать нужные потребительские ценности, которыми он не обладал раньше. Необходимо здраво оценивать возможности выхода с новым продуктом на уже занятый производителями рынок.

Эффективный путь выхода на рынок – поиск «маргинальных ниш»: групп потребителей, остро нуждающихся именно в особых свойствах нового продукта. При описании потребительских ценностей продукта важно структурировать их по отдельным целевым группам потребительской цепочки – строить так называемую «стратегическую канву»: определяем потребности каждого из них, определяем «нашего» потребителя, связываем потребительские ценности с характеристиками объекта и т.д. Пример такой стратегической канвы показан на Рис. 14.3.4 (пример взят из презентации менеджера по стратегическому развитию компании Intel в России И.О.Одинцова).

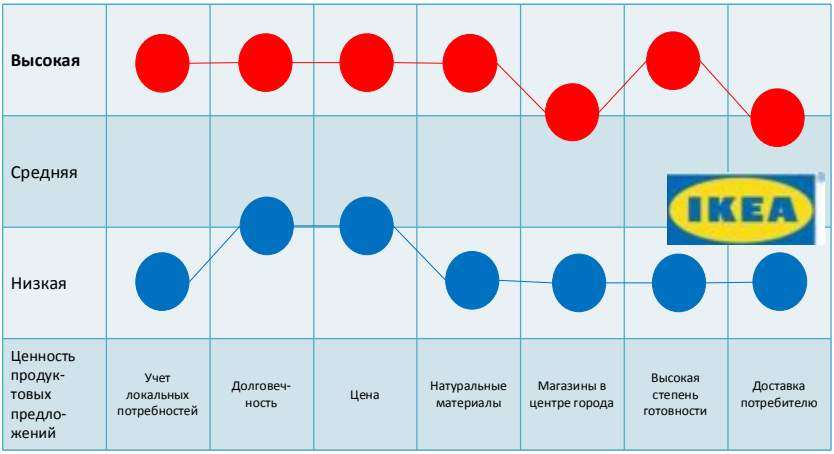


Рис. 14.3.4. Построение стратегической канвы на примере компании Ikea.

Варианты описания преимущества представляемого продукта:
1. через формулирование преодоленного противоречия, мешающего обеспечить дальнейшее развитие конкурирующих объектов;
2. через рост эффективности полезных свойств и характеристик по сравнению с объектами – аналогами;
3. через сравнение потенциалов дальнейшего развития объектов конкурентов и выявление преимуществ предлагаемого объекта по важной характеристике
4. через предъявление новой удовлетворенной потребности, проявляющейся в генеральном тренде (но не реализуемой пока в данном классе объектов).

Формулирование технического или физического противоречия, мешающего обеспечить гармоничное развитие конкурирующих объектов и демонстрацию способа его преодоления в предлагаемом объекте, технологии выгодно выделяет продуктовое предложение, демонстрируя его отличительные свойства. Такая демонстрация, особенно подкрепленная показом возможности патентной защиты, способна ярко показать инвесторам плюсы и отличие проекта от прочих.

Кроме этого, отличительные свойства следует продемонстрировать с помощью бенчмаркинга и функционально-стоимостного анализа. Бенчмаркинг – это процедура сравнения сложных систем по многим факторам с целью выявления лидирующей системы. Для сравнения принимаются продукты или системы с одинаковыми главными функциями (системы – конкуренты). В таблицу для сравнения следует вносить не только системы, реально существующие на рынке, но и находящиеся в стадии разработки, описанные в патентах и других литературных источниках, т.е. условно существующие.

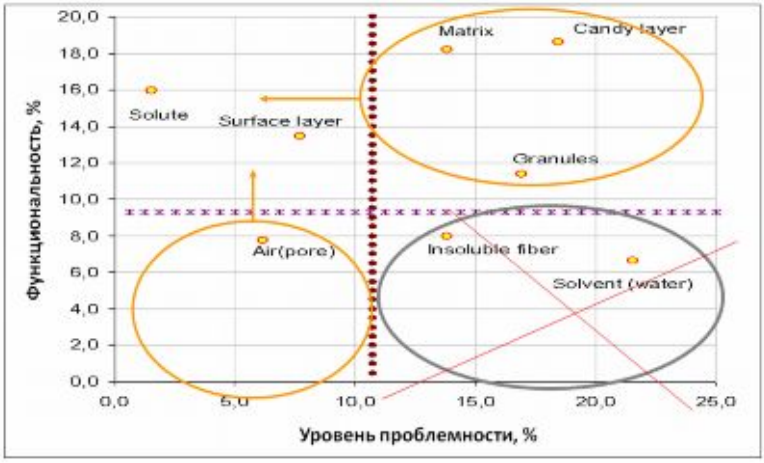


Рис. 14.3.5. Поле тримминга.

Для оценки достоинств продукта можно применять и тримминг. Тримминг – инструмент выявления элементов системы с наихудшим отношением функциональности к проблемной значимости с целью их последующего сокращения. Формально тримминг-фактор можно записать в виде простого соотношения: T = F/(P+C), где T – тримминг фактор; F – функциональная значимость (%), P – проблемная значимость (%), C – затратная значимость (%).

Представление продуктового предложение – очень важный, но не единственный шаг в организации стартапа. Если инвестор оценил и принял предложение, согласился инвестировать средства в

проект, то следующим шагом будет планомерное потсроение самой компании, в которой будет разворачиваться стартап.

Современная компания – сложная система, живущая и развивающаяся по законам сложных систем. Она начинается с миссии, концепции, слогана, целей, задач, стратегии достижения целей через решение задач, бизнес-модель достижения результата, имеет несколько уровней функционирования и управления (Рис. 14.3.6 и 14.3.7).

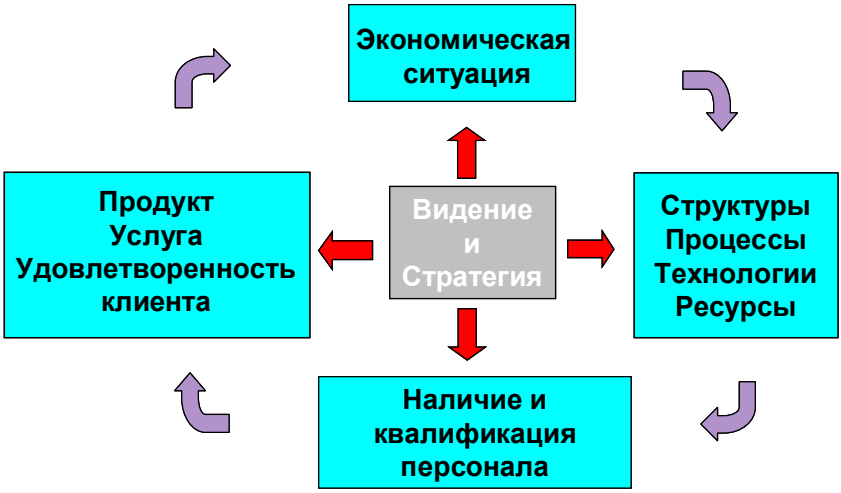


Рис.14.3.6. Концептуальное поле компании.

В широком понимании миссия рассматривается как философия и предназначение компании в контексте её существования и развития Миссия компании – это база для последовательного формирования целей, стратегий, функций, процессов, должностных инструкций и т.д. Миссия представляет собой результат позиционирования компании среди других участников рынка. Приведем два определения, которые закрепляют верхний и нижний уровень миссии.

«Миссия – стратегическая (генеральная) цель, выражающая смысл существования, общепризнанное предназначение организации. Это роль, которую предприятие хочет играть в обществе» (Л.Гительман «Преобразующий менеджмент»).

«Миссия (предназначение) организации – ответ на вопрос, в чем заключается деятельность компании, и чем она намерена заниматься» (И.Мазур, А.Шапиро «Реструктуризация предприятий и компаний»).

Рис. 14.3.7. Онтологическое поле компании.

Миссия как основная деловая концепция формулируется чаще всего на базе восьми ключевых вопросов:

- что получит поребитель в части удовлетворения своих потребностей?
- кто, для чего и как может выступать в качестве партнера компании?
- на какой основе предполагается строить отношения с конкурентами (какова, в частности, готовность пойти на временные компромиссы)?
- что получит собственник и акционеры от бизнеса?
- что получат от бизнеса и компании менеджеры?
- что получит от компании персонал?
- в чем может заключаться сотрудничество с общественными организациями?

- как будут строиться отношения компании с государством (в частности возможное участие в поддержке государственных программ)?

В конечном счете, полнота описания и детальность проработки миссии – бесспорный признак зрелости компании и критерий качества ее бизнеса. Профессионально разработанная миссия – это один из определяющих факторов конкурентоспособности компании: способность лучше других находить компромисс собственных интересов с интересами всех участников внешнего окружения и внутренней организации компании!

Следующая задача первостепенной важности – определиться с бизнес-моделью компании и её выражением в виде толкового бизнес-плана.

По определению Haas Business School ««Бизнес- модель – это совокупность определяющих бизнес-решений, а также уступок и компромиссов, задействованных компанией для извлечения прибыли». Бизнес-модель включает в себя набор механизмов, связывающих все компоненты бизнеса и определяющих как компания будет получать доход и прибыль. Бизнес-модель включает в себя множество компонентов: идентификацию клиентов, способы достижения конкурентного преимущества, процесс создания «value added» для клиентов, организацию эффективных бизнес-процессов, методы привлечения средств, способы завоевания и удержания клиентов и многое другое.

Выделяют четыре основных категории определяющих бизнес-решений и компромиссов:

- необходимый размер инвестиций
- источники затрат
- источники дохода;
- критические факторы успеха.

Отметим, что практически невозможно оценивать компоненты бизнес-модели независимо друг от друга. Основным отличием успешной бизнес-модели от неудачной является то, как построены взаимосвязи между всеми её компонентами.

Построение адекватного бизнес-плана – особая задача, тесно связанная как с организационно-функциональной структурой

будущего предприятия так, и с его бизнес-моделью и её экономической реализацией. Это выделенная область знания, существет достаточно много книг и учебных пособий по построению бизнес-планов (примеры можно посмотреть на сайте http://www.probp.ru/) [10]. Поэтому конспективно выделим несколько существенных моментов.

Целью разработки бизнес-плана является не только получение необходимого кредита, но в первую очередь – это планирование деятельности организуемой компании на ближайший и отдаленных периоды в соответствии с потребностями рынка и возможностями получения необходимых ресурсов. Бизнес-план помогает предпринимателю решить следующие основные задачи:

- определить конкретные направления деятельности компании, целевые рынки и место компании на этих рынках;
- сформулировать долговременные и краткосрочные цели фирмы, стратегию и тактику их достижения; определить лиц, ответственных за реализацию стратегии;
- выбрать состав и определить показатели товаров и услуг, которые будут предложены потребителям; оценить производственные и торговые издержки по их созданию и реализации;
- выявить соответствие имеющихся участников проекта, условий мотивации их труда предъявляемым требованиям для достижения поставленных целей;
- определить состав маркетинговых мероприятий по изучению рынка, рекламе, стимулированию продаж, ценообразованию, каналам сбыта и др.;
- оценить соответствие имеющихся финансовых и материальных ресурсов возможностям достижения поставленных целей;
- предусмотреть «подводные камни», которые могут помешать практическому выполнению бизнес-плана.

Таким образом, бизнес-план является рабочим инструментом для создания, функционирования и развития компании:

- философия бизнеса и способ достижения цели;
- образ создаваемой фирмы или стратегия развития существующей компании;
- условие нахождения партнеров или инвесторов;

- обоснованная оценка эффективности плана экономического замысла;
- основа инвестиционного проекта и условие для принятия решения о капиталовложении;
- путь от формирования бизнес идеи до получения прибыли;
- поисковая научно-исследовательская и проектная работа;
- программа страхования бизнеса от возможных рисков;
- эффективное средство продвижения бизнес идеи на рынок интеллектуальной собственности;
- лучший способ формирования имиджа фирмы;
- окно для внешнеэкономической деятельности;
- незаменимый метод анализа возможных ошибок и способ их предотвращения;
- универсальные возможности контроля, корректировки и мониторинг хода реализации бизнеса;
- реальная возможность использования программных, телекоммуникационных средств, а также выхода в Интернет;
- метод мотивации команды на высокие результаты.

Пренебрегая составлением бизнес-плана, предприниматель может оказаться не готовым к тем проблемам, которые ждут его на пути к успеху. А чаще всего это заканчивается плачевно как для него, так и для бизнеса, которым он занимается. Поэтому лучше не пожалеть времени и серьезно заняться бизнес-планированием. Документированное оформление бизнес-плана имеет очень существенное значение для организации работ по его выполнению. Не следует пренебрегать составлением бизнес-плана даже в условиях, когда ситуация на рынке меняется довольно быстро и часть запланированных мероприятий остаётся нереализованной.

Отметим, что не существует конкретного стандарта на разработку бизнес-плана из-за различия целей бизнеса и бесконечного множества вариаций среды, в которой он действует. Следовательно, требуются навыки и усидчивость, чтобы описать 3-х или 5-ти летнюю перспективу развития бизнеса. Степень детализации бизнес-плана соответствует целям фирмы и не включает ничего лишнего. Для привлечения внимания экспертов бизнес-план представляется интересным и легко воспринимаемым.

При обосновании прогнозов используется максимальное количество подтверждающих фактов для того, чтобы прогнозы были реалистичными.

Каждая компания строит свой бизнес-план, ниже приводится его основное содержание:

- Резюме
- Описание продукта или услуги
- Исследование и анализ рынка
- План маркетинга и организации продаж
- План дизайна и разработки продукта
- Производственный и операционный план
- Организационный план
- Интеллектуальная собственность и нематериальные активы
- Риски, проблемы, предположения
- Финансовый план
- Продуктовое предложение в целом

Завершить представление комплекта документов по бизнес-плану нужно следующим образом: резюме проекта (1-2 страницы), презентация для инвестора (12-12 слайдов), описание линейки продуктов (1-4 страницы), более подробная презентация для партнеров и возможных дистрибьюторов (15-20 слайдов). Укажем на типичные ошибки представления бизнес-плана:

- нет анализа аналогов продукта и конкурентов (особенно зарубежных);
- не проработана бизнес-модель;
- нет «портрета» организаторов и команды предприятия;
- дано описание технологии, а не новой потребительской ценности;
- перегруженность техническими деталями;
- отсутствует понимание защиты интеллектуальной собственности»
- поверхностное описание действий по реализации проекта, продажам, выводу на рынок и продвижению продукта.

Какими бы мотивами не руководствовался предприниматель, начиная деятельность по организации стартапа, алгоритм его реализации должен включать необходимые этапы и артефакты, о

которых говорилось выше. При всём многообразии форм предпринимательства существуют ключевые положения, применимые практически во всех областях коммерческой деятельности, необходимые для того, чтобы своевременно подготовиться, обойти потенциальные трудности и опасности, тем самым уменьшить риск в достижении поставленных целей. Разработка стратегии и тактики производственно-хозяйственной деятельности на основе миссии, стратегии, бизнес-модели и бизнес-плана компании является важнейшим условиями для построения эффективного бизнеса.

В заключение приведем замечательные слова одного из основателей корпорации Intel Роберта Нойса:

«Не идите на поводу у истории. Проявите инициативу и сделайте что-нибудь удивительное»

Robert Noyce, Intel Founder

14.4. Лабораторная работа № 12 «Размещение приложения в Intel AppUp»

14.4.1. Цель лабораторной работы

Научиться подготавливать свое приложение для размещения в Intel® AppUp и загружать его в систему Intel AppUp developer program.

14.4.2. Введение

Идеальный случай, когда придуманное для себя интересное приложение можно клонировать для других. Появляется сверхприбыль и т.п. Если мыслить предпринимательски, то, придумывая новые приложения, надо думать о том, как их сделать доступными для других, сохраняя при этом свои права.

Intel AppUp – это новый сервис, предоставляющий пользователям каталог приложений для нетбуков и персональных компьютеров с возможностью покупки и загрузки.

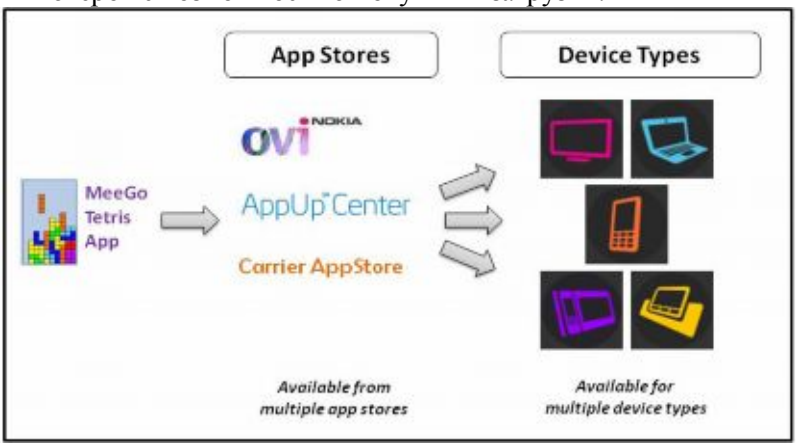


Рис. 14.5.1.

Intel AppUp помогает разработчику многократно клонировать для других придуманное им интересное приложение. Можно не только продавать свои приложения, но и получать прибыль с разработок, в которых применяется ваш код. В этой лабораторной работе будет показано, как подготовить приложение для загрузки в систему и описаны все стадии регистрации в Intel AppUp developer program для отправки заявления на публикацию своего приложения в Intel AppUp.

14.4.3. Инструкция по выполнению лабораторной работы

14.4.3.1. Регистрация в Intel AppUp developer program

Для того чтобы стать разработчиком приложений Intel AppUp необходимо зарегистрироваться на официальном сайте Intel AppUp для разработчиков. Для этого переходим по ссылке http://appdeveloper.intel.com (Рис. 14.5.2):

Рис.14.5.2.

Далее вы можете ознакомиться с сайтом, узнать о возможностях, о сообществе разработчиков о том как работать с сайтом. Для регистрации перейдём по ссылке Join the program, которая расположена в синем поле под название SELL (Рис. 14.5.3):

Рис. 14.5.3.

В отрывшейся вкладке появится таблица, в которой описаны возможности трёх различных аккаунтов. Для того, чтобы создавать

свои приложения, необходимо зарегистрироваться как Program Members. Сейчас регистрация в Intel AppUpSM в качестве разработчика бесплатна. В дальнейшем такая регистрация будет стоить 99$ в год.

Нажимаем «Join Now» и переходим на вкладку регистрации и получения ID. Заполняем данные о себе и нажимаем «Get an ID». В появившемся окне нажимаем «Next». Появится сообщение о проверке введенного вами адреса электронной почты, нажимаем первое «GO». Через некоторое время на введенный вами e-mail придет проверочный код, который необходимо ввести в сообщение о проверке и нажать второе «GO».

После в открывшемся окне нажимаем «Next» и попадаем во вкладку создания новой организации. Здесь необходимо указать название своей организации, например «My Home». Нажимаем «GO». Далее понадобится заполнить данные о своей организации (можно ввести всё тоже самое, что и при получении ID). Нажимаем «Next».

Теперь потребуется информация об адресе вашего счёта в PayPal. Intel AppUp все денежные переводы осуществляет через PayPal, так что если у вас нет счёта там, то создайте его перейдя по ссылке https://www.paypal.com/, и далее «Sign in». После заполнения адреса PayPal нажимаем «Next». Далее читаем правовое соглашение, ставим галочку о согласии и нажимаем «Next», а затем «Done». На этом регистрация закончена, можно ещё раз проверить данные об организации и приступать к подготовке приложения.

14.4.3.2. Intel AppUp SDK

Для того чтобы ваше приложение было принято в систему Intel AppUp необходимо интегрировать его с Intel AppUp SDK. Для начала нужно скачать и установить Intel AppUp SDK.

1. Переходим по ссылке http://appdeveloper.intel.com/en-us/develop и нажимаем «Download SDK». Здесь мы видим несколько вариантов SDK. Скачиваем файл под нужную вам ОС и в удобном для вас виде. (Не забудьте скачать плагины, если потребуется).
2. Установливаем Intel AppUp SDK. Для этого

- в Windows просто открываем скаченный пакет .MSI и устанавливаем программу. Далее устанавливаем плагин, открывая нужный пакет .MSI.
- В ОС Moblin/MeeGo необходимо перейти в папку, куда записан файл с программой, и выполнить:
  - для deb-пакета:

```
sudo dpkg -i adpcore-0.9.1-1.i386.deb
```

  - для rpm-пакета:

```
sudo rpm -i adpcore-0.9.1-1.i386.rpm
```

  При отсутствии недостающих пакетов необходимых для установки, воспользоваться менеджером пакетов, например apt-get.

3. Далее SDK надо инициализировать, вызвав

```
ADP_Initialize();
```

   В результате возможны варианты: ADP_SUCCESS; ADP_FAILURE; ADP_NOT_AVAILABLE; ADP_INCOMPATIBLE_VERSION; ADP_TIMEOUT.
   При успешном первом исходе возможно продолжение работы.
4. Для авторизации вашего приложения необходимо получить производственный GUID (Global Unique Identifier). Для этого необходимо в системе Intel AppUp Developer Program зайти на панель управления «My dashboard» и нажать там на «Start A New Application». В появившемся окне необходимо ввести название своего приложения. В поле Application Information вы увидите графу GUID, где расположен уникальный ID вашего приложения.

14.4.3.3. Подготовка приложения

1. Определите, авторизована ли машина запускать ваше приложение (или приложение запускать вашу компоненту)

```
#include <stdio.h>
#include "adpcore.h"
int main( int argc, char* argv[] )
{
      ADP_RET_CODE ret_code;
```

```
      // Please use the application GUID obtained
from the Intel Atom Developers Portal or a
ADP_DEBUG_APPLICATIONID
      const ADP_APPLICATIONID myApplicationID = {{
0x00000000,0x11111111,0x11111111,0x11111111}};

      if ((ret_code = ADP_Initialize()) !=
ADP_SUCCESS ){
            printf( "ERROR: exiting" );
            exit( -1 );
      }
      if (( ret_code = ADP_IsAuthorized(
myApplicationId )) == ADP_AUTHORIZED )
            printf( "Hello World" );
      else
            printf( "Not authorized to run" );
      exit 0;
}
```

2. Отладка

```
int main( int argc, char* argv[] )
{
ADP_RET_CODE ret_code;
const ADP_APPLICATIONID myApplicationID =
ADP_DEBUG_APPLICATIONID;
...
```

Стартуйте сервис эмуляции

3. Собирайте статистику использования

```
...
// Record Application start
ret_code = ADP_ApplicationBeginEvent( );
//Core application code
...
// Record Application end
ret_code = ADP_ApplicationEndEvent( );
exit(0); // Application exit
```

4. Используйте возможность сбора отчётов о сбоях

```
void SampleCrashHandler(int signal)
{
...
```

```
      response = ADP_ReportCrash( module, lineNumber,
message, category, errorData, errorDataSize);
...
}
```

14.4.3.4. Загрузка приложения

Вернемся обратно к системе Intel AppUp developer program для загрузки приложения в Intel AppUp.

1. Заходим на страничку http://appdeveloper.intel.com/en-us/contest/contest-entry.
   Если Вы не авторизованы, то сначала появится страница авторизации.
2. Далее необходимо заполнить все поля с данными о вашем приложении:
   - Имя издателя
   - Название приложения (ваш уникальный ID)
   - Иконка вашего приложения (не менее 100*100 pix, в формате *.png, в имени файла не допускаются русские буквы)
   - Слоган
   - Краткое (max. 300 п.з.) и полное (max. 4000 п.з.) описание вашего приложения
   - Скриншоты (не менее одного, строго заданного размера 820*480 pix, *.gif, *.jpg или *.png, в имени файла также не допускаются русские буквы)
   - Важные данные о версии (если есть)
   - E-mail для поддержки приложения

   После заполнения нажимаем Next.
3. На следующей вкладке необходимо заполнить графу «цена приложения», и, нажав Next, перейти к заполнению информации о требованиях к аппаратной платформе и выбрать поддерживаемую вашим приложением операционную систему. Также необходимо заполнить информацию об аппаратной платформе на которой вы тестировали своё приложение и дополнительные сведения о приложении (если есть) для бета-тестеров команды Intel.

4. Далее загружаем свой пакет приложения и отправляем на проверку в команду Intel.

Ждём ответа проверки, следим за статусом своего приложения. Приложения, которые прошли проверку, потом выкладываются в Intel AppUp. Дальше остаётся следить за рейтингом и продажами своего приложения, зарабатывать деньги и придумывать новые интересные приложения.

14.4.4. Задания для самостоятельной работы

1. Придумайте свое приложение и разместите его в Intel AppUp.

14.5. Выводы

В лекции анализируются основные трудности коммерциализации идей в виде приложений для мобильных устройств, даются рекомендации по созданию стартапов и вместе с лабораторной работой объясняются преимущества и возможности Intel AppUp developer program.

14.6. Контрольные вопросы

1) Чем предпринимательская деятельность отличается от бизнес-деятельности?
2) Кто такой предприниматель и чем он отличается от бизнесмена?
3) Что такое инновационный проект и в чем заключается его отличие от не инновационного?
4) Может ли не инновационный продукт быть высокотехнологичным и хорошо продаваемым? И почему?
5) Из каких этапов состоит жизненный цикл инновационного продукта?
6) Чем определяется срок жизни инновационного продукта на рынке?
7) Каким путем лучше всего проводить коммерциализацию продукта?

8) С каких шагов следует начинать подготовку стартапа компании?
9) В чем состоит базовая суть продуктового предложения?
10) В какой системе координат следует описывать продуктовое предложение?
11) Что оценивает инвестор в первую очередь при подаче заявки на поддержку разработки продукта?
12) Какие базовые вопросы прежде всего ставит экспертиза при рассмотрении продуктового предложения?
13) Каким образом решаются вопросы интеллектуальной собственности при подготовке стартапа и выводе продукта на рынок?
14) Для чего необходим прототип продукта или его опытный образец?
15) Какие факторы рассматривают эксперты при оценке команды проекта?
16) Необходимость какого условия оценивается в первую очередь при экспертной проработке проекта?
17) Что такое бизнес-модель предприятия и кто занимается разработкой бизнес-модели?
18) С какой первоочередной целью разрабатывается слоган компании? Какую смысловую нагрузку он несёт?
19) Для чего создается бизнес-план и кто его читает?
20) Какие важные разделы включает в себя бизнес-план?

Список литературы

1. Dollinger M., Entreprenership: Strategies and Resource. Prentice Hall, 2003.
2. Граничин О.Н., Кияев В.И. Информационные технологии управления. М.: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2008.
3. Druker P. Innovation and Entrepreneurship. Harper Collins, 1985.
4. Лаптев Г.Д., Ладионенко М.А. и др. Компетентностный подход и роль дизайн-мышления в обучении инновационному предпринимательству. – М.: ТЕИС, 2010.
5. http://appdeveloper.intel.com/en-us/article/program-overview

15. Заключение

Основные итоги изучения курса и планы на будущее.

Уважаемые читатели и слушатели, вот и подошел к концу наш полугодовой вводный курс о том, как разрабатывать приложения на платформе Atom/MeeGo. Не смотря 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.