УДК 004:37

QT КАК СРЕДСТВО КРОССПЛАТФОМЕННОЙ РАЗРАБОТКИ

Лаврик А.В., Кутецкий Д.В.
Лаборатория разработки и внедрения педагогических программных средств при НИИ ИТ Херсонского Государственного Университета.

В данной статье рассматриваются возможности инструментарий разработки кроссплатформенного ПО Qt, и производится их сравнение с возможностями Java библиотек.

Ключевые слова: кроссплатформенность, Java, Qt, сравнение.

Вступление

Бурное развитие компьютерных технологий предоставляет всё больший спектр возможностей конечным пользователям продуктов, однако зачастую создаёт проблемы для их разработчиков. Одну из довольно острых проблем порождает обилие разного рода программных платформ — Windows, Unix/Linux, Mac OS, множество платформ мобильных устройств. Разработка отдельных версий программного обеспечения для разных платформ является трудоёмкой и давно признана экономически нецелесообразной. Вместо этого популярность приобрела концепция т. н. кросплатформенных приложений т. е. требующих минимальной переделки под различные платформы, или не требующих её вообще [1].

Одним из набирающих популярность кросплатформенных средств является Qt — высокоуровневый инструментарий разработки программ, разработанный фирмой Trolltech (ныне Qt Technologies, входит в Nokia Corporation). Qt написан на языке программирования C++ и ориентирован главным образом на него, но поддерживает «привязки» (адаптеры) к другим языкам программирования — Python (PyQt), Ruby (QtRuby), Java (Qt Jambi), PHP (PHP-Qt) и др[1, 2].

Данная статья имеет целью рассмотреть возможности разработки десктопных приложений предоставляемые инструментарием Qt версии от 4.0, в котором в полной мере реализованы мощные механизмы создания кроссплатформенного ПО. Также будет произведено сравнение Qt с другим популярным средством разработки — Java от компании Sun.

Лицензирование

На сегодняшний день Qt поставляется в двух вариантах: коммерческом и с открытым исходным кодом. Свободная версия может использоваться под лицензиеями GNU General Public License (GNU GPL) или GNU Lesser General Public License (GNU LGPL). Первая даёт права доступа к исходному коду, а также копирования, модифицирования и распространения библиотеки Qt, но требует распостранения вашего приложения на тех же условиях. GNU LGPL позволяет динамически связывать с библиотекой Qt программу под любой лицензией, т.е. исходный код приложения может быть закрыт, возможно распространение бинарных файлов на коммерческой основе и т.д. Коммерческая лицензия позволяет распространять программный продукт на ваших условиях, кроме того, вы получаете доступ к службе поддержки Qt Software и обновлениям. Возможен переход от коммерческой лицензии к лицензии GNU LGPL или GNU GPL, однако перевести проект, начатый с использованием GNU LGPL или GNU GPL версии Qt, на коммерческую версию Qt нельзя.[1,2,3]

Архитектура

Qt включает в себя множество классов от элементов графического интерфейса до классов для работы с БД, сетью, XML. Qt полностью объектно-ориентирован, поддерживает компонентное программирование и легко расширяется. С версии 4 делится на следующие логические модули (и бинарные библиотеки — .dll для Windows и .so для Unix) [4,5].

- QtCore — базовые классы, используемые другими модулями и программами пользователя (для создания консольных приложений);
- QtGui — содержит огромный (порядка нескольких сотен) набор классов элементов графического интерфейса пользователя;
- QtNetwork — для работы с протоколами FTP и HTTP предоставляются классы QFtp и QHttp, основаные на низкоуровневом классе QSocket, реализующем представление сокетов TCP. Протокол TCP работает в терминах потоков данных, передаваемых между узлами сети. Класс QSocket, в свою очередь, реализован поверх QSocketDevice — тонкой «обёртки» вокруг платформо-зависимого сетевого API операционной системы. Класс QSocketDevice поддерживает протоколы TCP и UDP.
- QtOpenGL — позволяет интегрировать трёхмерную графику OpenGL в десктоп-приложения (как видно из приложения Google Earth, которое разрабатывалось на Qt, возможно и создание аналогичного Direct3D виджета);
- QtSql — интерфейс для работы с базами данных SQL; Qt включает «родные» драйвера для Oracle, Microsoft SQL Server, Sybase Adaptive Server, IBM DB2, PostgreSQL, MySQL и ODBC-совместимых баз данных. Qt включает специфичные для баз данных виджеты, а также поддерживает расширение для работы с базами данных любых встроенных или отдельно написанных виджетов. [6]
- QtXml — работа с XML. Qt поддерживает два различных API:
  - SAX (Simple API for XML — простейший прикладной интерфейс для работы с XML) — используется для выполнения синтаксического анализа методом обработки событий разбора прямо в приложении, с помощью виртуальных функций.
  - DOM (Document Object Model — объектная модель представления документов) — преобразует XML-документ в древовидную структуру, в результате приложение получает возможность навигации по ней.

Дополнительные классы

1. QtAssistant — поддержка для файлов справки;
2. Qt3Support — поддержка совместимости с Qt3;
3. QAxContainer — контейнер ActiveX (только под Windows);
4. QAxServer — сервер ActiveX (только под Windows);
5. QtDesigner — классы Дизайнера — редактора GUI.

Данные модули легко подключаемые и свободно переносимые (разделение на физические файлы), т.е., если в приложении используются классы для работы с сетью и XML то кроме обязательных QtCore и QtGui переносятся лишь файлы QtXml.(dll/so) и QtNetwork.(dll/so) (), а не вся библиотека Qt [5,7].

Отличительные черты инструментария Qt

1. Для работы необходимы лишь библиотеки Qt и компилятор C++.
2. Meta Object Compiler (MOC) — система предварительной обработки исходного кода на Qt в стандартный синтаксис C++. Главным нововведением являются *слоты* и *сигналы* — средства призванные заменить callback-функции.
3. Менеджеры компоновки вместо абсолютного размещения виджетов.
4. Абстрагирование от конкретной ОС. Qt выступает абстрактной прослойкой между ПО и ОС, скрывая от разработчика ОС-зависимые механизмы реализации каких-либо возможностей; вместо них используются кроссплатформенные обёртки.
5. Встроенная поддержка Unicode и локализации.
6. «Родной вид» приложения. По умолчанию стилем приложения является стиль операционной системы, в которой оно компилировалось.
7. Иерархические и настраиваемые объектные деревья, организующие принадлежность объектов естественным образом.

8. Защищённые указатели QGuardedPtr, автоматически принимающие значение NULL при уничтожении соответствующего объекта, в отличие от обычных указателей C++, которые стают неопределёнными.
9. Удобная документация, частично доступная и на русском языке

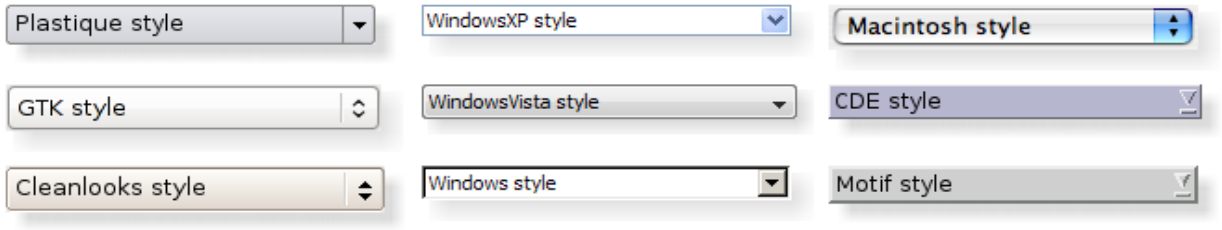

Иллюстрация 1: Вид элемента «Combo Box» в различных операционных системах

Поддерживаемые платформы

Существуют сборки Qt для Microsoft Windows, Unix/Linux с графической подсистемой X11, Mac OS X, Microsoft Windows CE, встраиваемых Linux-систем и платформы S60, ведётся портирование на HaikuOS [1,3,5].

Сравнение с другими средствами разработки

Проведём сравнение Qt с другим популярным кроссплатформенным средством разработки – библиотекой Java Swing (как библиотеки Qt/Swing, так и языки инструментария – Java/C++) по следующим критериям:

1. Кроссплатформенность.
2. Скорость разработки ПО.
3. Производительность приложений.
4. Эффективность использования памяти
5. Сравнение библиотек Qt и Java Swing/AWT

Кроссплатформенность

Java и Qt используют различные подходы к кросплатформенности:

1. Java: Компилятор генерирует байт-код, непосредственно исполняемый не процессором, а виртуальной машиной Java (JVM). В свою очередь, JVM исполняется процессором. Таким образом, выполнение Java-программ осуществляется не быстрыми аппаратными средствами, а более медленной программной эмуляцией. Для повышения производительности их работы разработаны «Just in Time» (JIT) компиляторы, но универсального решения проблемы не существует.
2. Qt: Система предварительной обработки исходного кода Meta Object Compiler (MOC) преобразует код в чистый C++, транслируемый компилятором в платформозависимый двоичный формат, исполняемый непосредственно процессором; таким образом, выполнение программы осуществляется аппаратными средствами.

Теоретически оба подхода обеспечивают кроссплатформенность. На практике виртуальные машины Java для различных платформ могут иметь существенные различия, не всегда обеспечивая совместимость, тогда как для обеспечения кроссплатформенности Qt-программ достаточно лишь библиотеки Qt и компилятора C++[8].

Скорость разработки ПО

Одно из главных преимуществ Java перед другими языками программирования — высокая продуктивность разработки ПО, в основном за счёт производительности работы приложения или эффективного использования памяти.

В Java специальные механизмы обеспечивают неявное освобождение неиспользуемой памяти — «сборку мусора» (в C/C++ это делается вручную). Она автоматически выполняется средой Java в ущерб производительности и эффективности использования памяти. Это

освобождает разработчика от утомительной задачи по слежению за освобождением памяти – главного источника ошибок в приложениях. По идее эта возможность языка должна значительно увеличить продуктивность работы по сравнению с C/C++.

Однако проводимые исследования показывает, что на практике сборка «мусора» и другие возможности Java не особо влияют на продуктивность программирования. Одна из классических моделей оценки программного обеспечения CoCoMo, предложенная Barry Boehm, предопределяет стоимость и сроки разработки программного продукта на основе стоимостных коэффициентов, которые учитывают такие факторы, как суммарный опыт программирования разработчика, опыт программирования на заданном языке, желаемая надёжность программы и т.д. Боем пишет, что независимо от уровня используемого языка, начальные трудозатраты всегда высокие. Подобная методика подсчета использовалась и исследованиях C.E.Walston и C.P.Felix, IBM, «Метод измерения и оценки программирования» (A method of programming measurement and estimation) .

Оба исследования проводились задолго до создания Java, но несмотря на это, они демонстрируют общий принцип: сложность языка программирования общего назначения по сравнению с другими аспектами, такими как квалификация разработчика, не оказывает существенного влияния на полную стоимость разработки проекта.

Существует такой способ определения продуктивности программирования как метод функциональных единиц (Function Point), разработанный Кэперс Джонс (Capers Jones). Функциональная единица – метрика ПО, зависящая только от его функциональности, а не от конкретной реализации. Она позволяет использовать в качестве критерия оценки продуктивности программирования число строчек кода, которые необходимы для обеспечения одной функциональной единицы, т.е. уровень языка определяет число функциональных единиц, которые можно создать за определенное время. Интересно, что обе величины: число строк кода на единицу функциональности и уровень языка одинаковы для обоих языков (уровень языка: C++ и Java – 6, C – 3.5; число строк кода на единицу функциональности: C++ и Java – 53, C – 91).

Можно заключить, что Java не обеспечивает большую производительность создания приложений, чем C++ [6,8].

Производительность работы приложений

По результатам тестов на производительность [4] было сделано заключение, что в силу использования виртуальной машины «Java-программы выполняются по крайней мере в 1.22 раза медленнее C/C++ программ». Однако опыт многих программистов показывает, что Java-программы выполняются в 2-3 раза медленнее, чем их C/C++ аналоги. На задачах, интенсивно использующих процессор, Java-программы отстают ещё сильнее.

Для программ с пользовательским графическим интерфейсом увеличение времени отклика интерфейса критичнее производительности программы. Проведенные исследования показывают, что пользователи более терпимы к задачам, работающим в течение нескольких минут, чем к программам, не реагирующим мгновенно на их действия, например, на нажатия кнопок. Было установлено, что программы со временем отклика больше, чем 0,7 секунды, ощущаются медленными. Мы вернёмся к этому при сравнении пользовательского графического интерфейса в программах Java и C++.

Возникает вопрос, стоит ли использовать программную реализацию виртуальной машины Java, если эту же функциональность может предоставить аппаратная часть? Разработчики языка Java предполагали, что вопрос низкой производительности будет решён с появлением доступной аппаратной реализации JVM в виде Java-процессоров. Однако последние до сих пор не получили широкого распространения [5, 7].

Эффективность использования памяти

Java и C++ используют различные подходы в управлении памятью. В C++ распределением и освобождением памяти полностью управляет программист. В результате

его забывчивости могут возникать «утечки памяти». Отдельно взятая утечка не критична, т.к. после завершения работы приложения ОС освобождает всю ранее использованную им память. Но при постоянных утечках памяти (например при периодическом повторении действия, приводящего к утечке), расход памяти приложением будет расти до полного её заполнения с последующим возможным отказом системы.

«Сборщик мусора» Java обеспечивает автоматическое освобождение неиспользуемой памяти. Вместе с распределением памяти программистом, JVM контролирует указатели на используемые блоки памяти. Если блок памяти больше не задействован, он может быть освобождён.

Сборка мусора довольно удобна, но за её использование надо расплачиваться большим потреблением памяти и низкой производительностью приложения. Программисты C++ могут (и должны) освобождать блоки памяти сразу после того, как они перестали нуждаться в них. В Java блоки не освобождаются до следующего вызова сборщика мусора, период работы которого зависят от реализации используемой Java-машины. Кроме большого расхода памяти процесс сборки мусора требует дополнительной мощности процессора, которая в результате становится недоступной приложению, что замедляет его работу. Поэтому вызовы сборщика мусора могут приводить к «зависанию» Java-программы на несколько секунд. Лучшие реализации Java-машин минимизируют такие торможения, но не устраняют их полностью.

При работе с внешними программами и устройствами (во время ввода/вывода или при взаимодействии с базой данных), желательно закрыть файл или соединение с базой данных сразу же после того, как они перестали быть нужны приложению. Благодаря деструкторам C++ это происходит сразу после вызова оператора delete. В Java закрытие произойдёт в следующем цикла работы сборщика мусора. В лучшем случае это приведёт к избыточной блокировке, в худшем – к нарушению целостности открытых ресурсов.

Использование программами Java больших блоков памяти особенно критично для встраиваемых устройств с ограниченными объёмами памяти. Главная причина, по которой сборка мусора является более дорогостоящей, чем непосредственное управление памятью программистом, – утрата информации. В C++ программе программист знает и местонахождение своих блоков памяти (сохраняя указатели на них), и когда они перестанут быть ему нужными. В Java-программе последняя информация недоступна для JVM (даже если она известна программисту), поэтому перебираются все блоки на предмет отсутствующих указателей. Для принудительного вызова сборщика мусора Java-программист может удалить все указатели на соответствующие блоки памяти. Но это потребует больше усилий, чем непосредственное управление памятью в C++; а во время сборки мусора JVM всё равно будет проверять все блоки памяти, чтобы освободить неиспользуемые[9].

С технической точки зрения сборка мусора возможна и в C++ программах. Существуют обеспечивающие это коммерческие программы и библиотеки. Но из-за перечисленных выше недостатков немногие C++ программисты используют их. Инструментарий Qt использует более эффективный подход для упрощения задачи управления памятью: при удалении объекта все зависящие от него объекты также автоматически удаляются. Однако ничто не мешает программисту по мере надобности самостоятельно удалять объекты.

Так как управление памятью в C/C++ обременительно для программиста, созданное с помощью них ПО обвиняется в нестабильной работе и подверженности ошибкам. Хотя некорректная работа с памятью в C/C++ может привести к более критичным ошибкам (в т.ч. аварийному завершению программы), хорошие знания, инструментарий и опыт могут значительно уменьшить связанный с этим риск. Изучению управления памятью должно уделяться достаточно внимания. Также существует большое число коммерческих и бесплатных инструментов, позволяющих программистам обеспечить отсутствие в программах ошибок при работе с памятью. Гибкая система управления памятью в C++

делает возможным создавать адаптированные для любого типа приложений профилировщики памяти.

Мы можем убедиться, что при сопоставимой продуктивности программирования C++ обеспечивает лучшую производительность работы и эффективность использования памяти приложениями, чем Java.

Сравнение AWT, Swing и Qt

Фреймворк AWT (Abstract Windowing Toolkit) поставляется вместе начиная с самой первой версии Java. Он использует родные для платформ компоненты GUI (т.е. Win32 API для Windows и библиотеку Motif для Unix), обеспечивая таким образом переносную обёртку. Т.е. внешний вид и поведение AWT-программ будет отличаться на различных платформах, потому что именно они занимаются отрисовкой и управлением компонентами GUI.

Затем AWT был дополнен инструментарием Swing. Swing использует AWT (и, следовательно, низкоуровневые библиотеки) лишь для базовых операций: создания прямоугольных окон, управления событиями и отрисовки графических примитивов. Всем остальным, включая отрисовку компонентов GUI, занимается Swing. Это решает проблему отличающегося внешнего вида и поведения приложений в различных ОС. Но из-за реализации Swing-инструментария средствами Java его производительность не слишком хороша. В результате Swing-программы притормаживают не только во время интенсивных вычислений, но и при отрисовке элементов пользовательского интерфейса. Хотя с ростом производительности оборудования эта ситуация постепенно улучшается, сложным пользовательским интерфейсам, созданным с помощью Swing, всегда будет свойственна медлительность.

При разработке инструментария Qt был использован тот же самый подход: низкоуровневые библиотеки используются только лишь для базовых операций, а отрисовкой элементов GUI занимается непосредственно Qt. Благодаря этому инструментарий Qt приобретает все преимущества Swing (например, схожесть поведения и внешнего вида приложений на различных платформах), но не имеет проблем с низкой производительностью, так как разработан на C++ и откомпилирован в машинный код. Интерфейс, созданный с помощью Qt, отличается быстрой работой, и, благодаря кешированию, может быть быстрее интерфейса, разработанного стандартными средствами [10]. Теоретически, оптимизированная не-Qt программа должна быть быстрее аналогичной Qt-программы; но на практике для оптимизации не-Qt программы потребуется больше усилий и мастерства, чем для создания оптимизированной Qt-программы.

И Qt, и Swing поддерживают технику стилей, позволяющей программам независимо от платформы использовать один из стилей интерфейса. Это становится возможным благодаря тому, что отрисовкой элементов GUI занимаются непосредственно Qt и Swing. Вместе с Qt поставляются стили, эмулирующие внешний вид Win32, Motif, MacOS X Aqua (в Macintosh-версии) и даже Swing-программ.

Выводы:

Мы рассмотрели платформу Qt и сравнили её с Java Swing, оценив пригодность обеих для разработки высокопроизводительных приложений с пользовательским графическим интерфейсом. В то время, как Java обеспечивает разработчикам сравнительно большую продуктивность программирования, платформа Qt предлагает лучшую производительность и эффективность использования памяти. Что касается библиотек, Swing и Qt, то очевидно, что худшая производительность Java-программ делает платформу Java Swing менее подходящей для разработки GUI-приложений, даже при одинаковом опыте программирования. В отличие от Swing, Qt не навязывает программисту парадигму программирования Model-View-Controller, поэтому в результате Qt-программы получаются более краткими.

Независимое научное исследование и практический опыт эксплуатации демонстрируют, что при разработке десктопного ПО использование Qt вместо Java вполне оправдано. Причины заключаются в низкой производительности и неэффективности использования памяти в Java (при использовании инструментария Swing) при практически

том же уровне продуктивности программирования. Java Swing подходит для разработки без или с ограниченной GUI-функциональностью. В целом связка C++/Qt является оптимальным решением для разработки GUI-приложений, т.к. предоставляет сравнимые возможности/скорость разработки при большей производительности приложений.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ж. Бланшет.Qt 4: Программирование GUI на C++. 2-е дополненное издание/ Ж. Бланшет, М. Саммерфилд. — М.: «КУДИЦ-ПРЕСС», 2008. — С. 736.
2. Макс Шлее Qt 4.5 Профессиональное программирование на C++/ Макс Шлее — СПб.: «БХВ-Петербург», 2010. — С. 896
3. Чеботарев А. Библиотека Qt 4. Создание прикладных приложений в среде Linux./ Чеботарев А — М.: «Диалектика», 2006. — С. 256.
4. Земсков Ю.В. Qt 4 на примерах./ Земсков Ю.В. — СПб.: «БХВ-Петербург», 2008. — С. 608.
5. http://doc.crossplatform.ru/ Режим доступа [http://doc.crossplatform.ru/qt/4.3.2/] Проверено [15.04.2010].
6. http://www.linuxcenter.ru. Режим доступа [http://www.linuxcenter.ru/lib/books/qt3/] Проверено [14.04.2010].
7. http://doc.crossplatform.ru/ Режим доступа [http://doc.crossplatform.ru/qt/4.3.2/] Проверено [14.04.2010].
8. http://ais.khstu.ru. Режим доступа [http://ais.khstu.ru/Reference/Qt/style-reference.html] Проверено [15.04.2010].
9. http://doc.crossplatform.ru. Режим доступа [http://doc.crossplatform.ru/qt/4.5.0/mainclasses.html]. Проверено [14.04.2010].
10. Lutz Prechelt. An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl / Lutz Prechelt - University of Karlsruhe, p. 134.

Рецензент: Львов М.С.