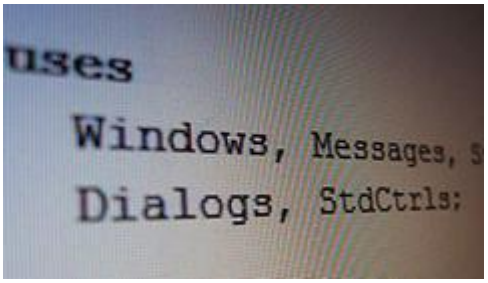
Java vs Delphi: мысли на границе чистовика и черновика

24 Мар 2016

Илья Манусов



Сравнение языков программирования сродни попыткам передать одним битом три состояния. Незабвенная Екатерина Логвиновна Ющенко, автор нескольких настольных книг по ныне раритетным и уже забытым языкам программирования, любила подначивать студентов вопросом: «Что лучше – *Fortran* или *Algol*?» Правильный ответ был таким: *Cobol*. Частенько, в зависимости от ситуации, названия языков менялись местами. Ирония Екатерины Логвиновны понятна – если одна система лучше во всех отношениях, нет смысла в существовании всех прочих. Но желанию разработчика небольшого проекта дистанцироваться от «конфликта парадигм» не суждено сбыться. Программист всегда стоит перед выбором технологии для достижения реальных целей.

О предмете обсуждения

Утилита PowerInfo, написанная в тестовой лаборатории «Компостера», своим появлением обязана среде разработки Lazarus и компилятору Free Pascal (FPC). Мгновенное создание и настройка объектов пользовательского интерфейса методами визуального программирования позволяет сосредоточиться исключительно на предметной области. Возможностей для декларирования сигнатур функций и задания уникальных идентификаторов GUID, оказалось достаточно, чтобы обеспечить вызов Power Management API и процедур IOCTL операционной системы, не прибегая к ассемблерным вставкам. Но по мере движения от инженерных релизов к статусу чистовика, мы обратили внимание и на недостатки, например размер выполняемого файла простейшего приложения составляет несколько мегабайт. Так родилась идея сравнения, обозначенного в заголовке статьи.

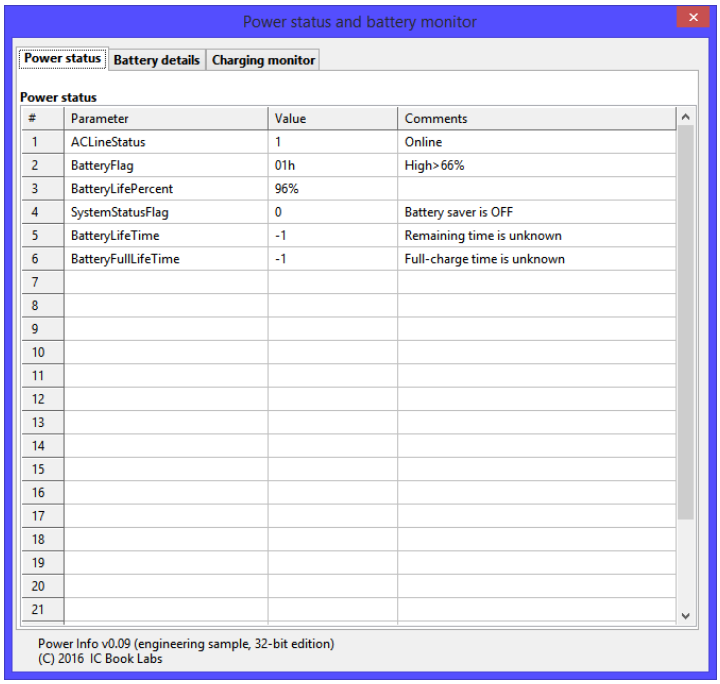


Рис.1 *Утилита Power Info разработки IC Book Labs. Экран информации Power Management подсистемы*

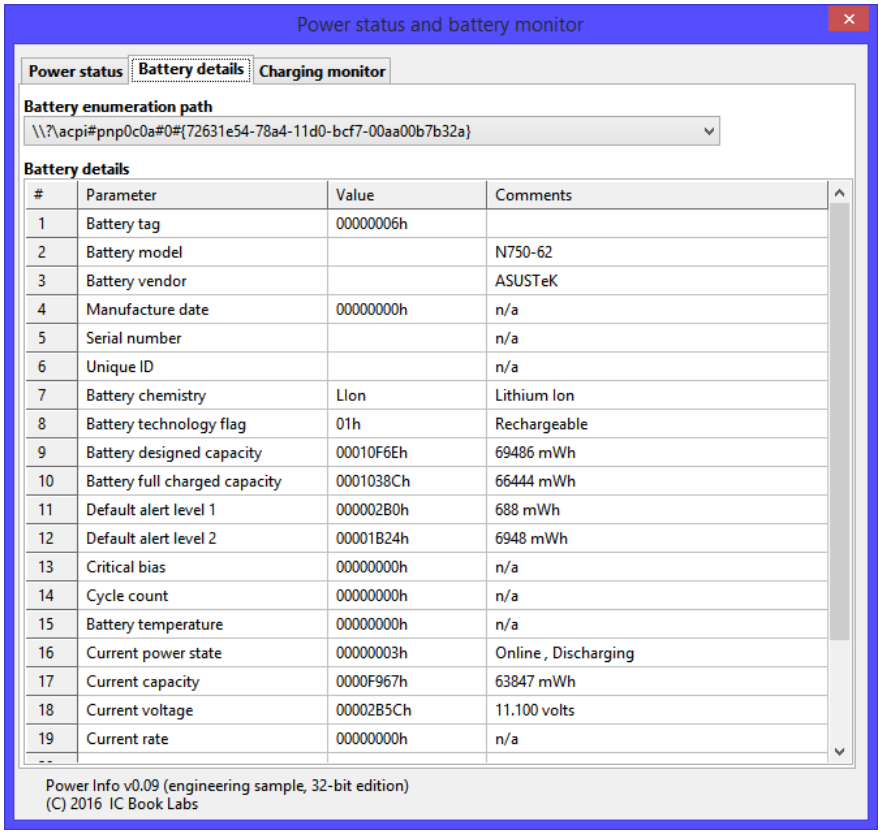


Рис.2 *Экран детальной информации подсистемы батарейного электропитания*

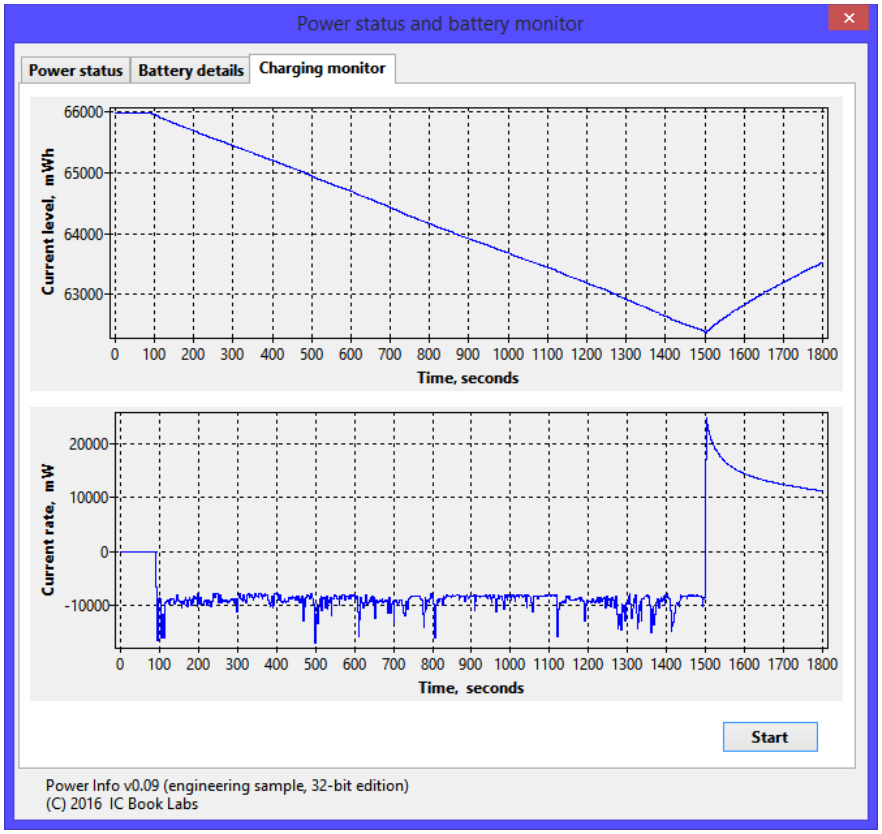


Рис.3 *Экран мониторинга состояния аккумуляторной батареи: верхний график соответствует текущему уровню заряда, нижний график соответствует зарядной мощности, которая положительна в случае заряда батареи и отрицательна при разряде (при работе ноутбука от батареи)*

Критерий 1: производительность

По субъективному мнению автора, код, от которого хотя бы в малейшей степени зависит быстродействие программы должен быть написан на *ассемблере*. Но так как это мнение *субъективное*, говорить будем совсем о другом. Опираясь на простейшие представления о виртуальных машинах и байт-коде, можно предположить, чтоJava-приложения должны работать значительно медленнее нативных, так как программная интерпретация данных (то есть выполнение Java байт-кода) медленнее, чем аппаратная интерпретация процессорных инструкций. Вместе с тем, в арсенале Java-машины имеется огромное количество хитростей и уловок, одной из которых является *динамическая трансляция*, то есть перевод в нативный код многократно выполняемого фрагмента. А так как в момент выполнения, о реальном поведении программного кода известно больше, чем в момент сборки, то и потенциальные возможности для оптимизации шире. В любом случае, принятию решения должен предшествовать эксперимент с конкретной реализацией JVM, так как наличие теоретической возможности еще не означает ее практическое использование.

Пересмотреть стереотипы о «неповоротливой» Java-машине, можно ознакомившись с ее *аппаратной* реализациейв рамках архитектуры ARM.

Критерий 2: размер выполняемого файла

Утилита *PowerInfo v0.09* (Рис. 1-3), выводящая две таблицы и один график на основании информации Power Management API, написанная на Free Pascal и собранная в среде разработки Lazarus, имеет размер около 3 мегабайт в варианте Win32 и около 4.5 мегабайт в варианте Win64. Реализация аналогичной функциональности на ассемблере займет менее 10 килобайт. Суммарный размер набора Java-классов, решающий задачу данного уровня сложности, не превысит отметку 40 килобайт.

Да, такое сравнение не объективно, поскольку Java-приложение требует загрузки операционной среды *JRE (JavaRuntime Environment)* достаточно внушительного размера и опирается на ее функции, а программа, собранная компилятором *Free Pascal*, является самодостаточной.

Здесь самое время вспомнить о тенденциях развития ОС для *гаджетов*, в которых JRE является неотъемлемой частью ОС, а не загружается ради запуска одной утилиты. Справедливости ради, отметим, что качество выполняемого кода, генерируемого Free Pascal, достаточно высокое, а причиной большого размера выполняемого файла, вероятно является подключение библиотечных модулей, независимо от факта их использования.

Критерий 3: удобство разработчика

Нам показалось очевидным, что для «макетирования» функциональности и мгновенного получения результата в условиях тестовой лаборатории, идеально подходят возможности визуального программирования в среде *Lazarus* в сочетании с компилятором *Free Pascal*. Но если вашему проекту суждено пересечь границу между черновиком и чистовиком, то такой подход уже не видится безупречным и одной из альтернатив может быть *Java*. Хотя вполне возможно, что использование этой технологии для написания информационно-диагностических утилит, покажется нетипичным. А возможно и не покажется, если портирование для ОС *Android* входит в ваши планы…

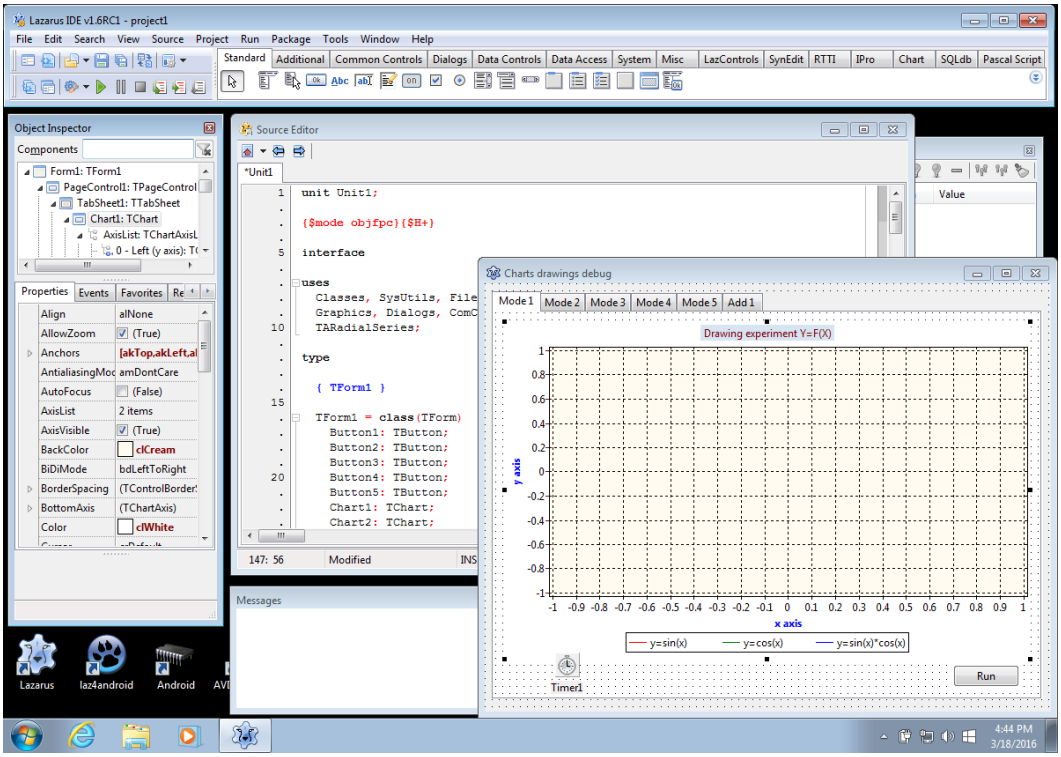


Рис.4 *Интегрированная среда разработки Lazarus, подключен компилятор Free Pascal и поддерживаются средства визуального программирования*

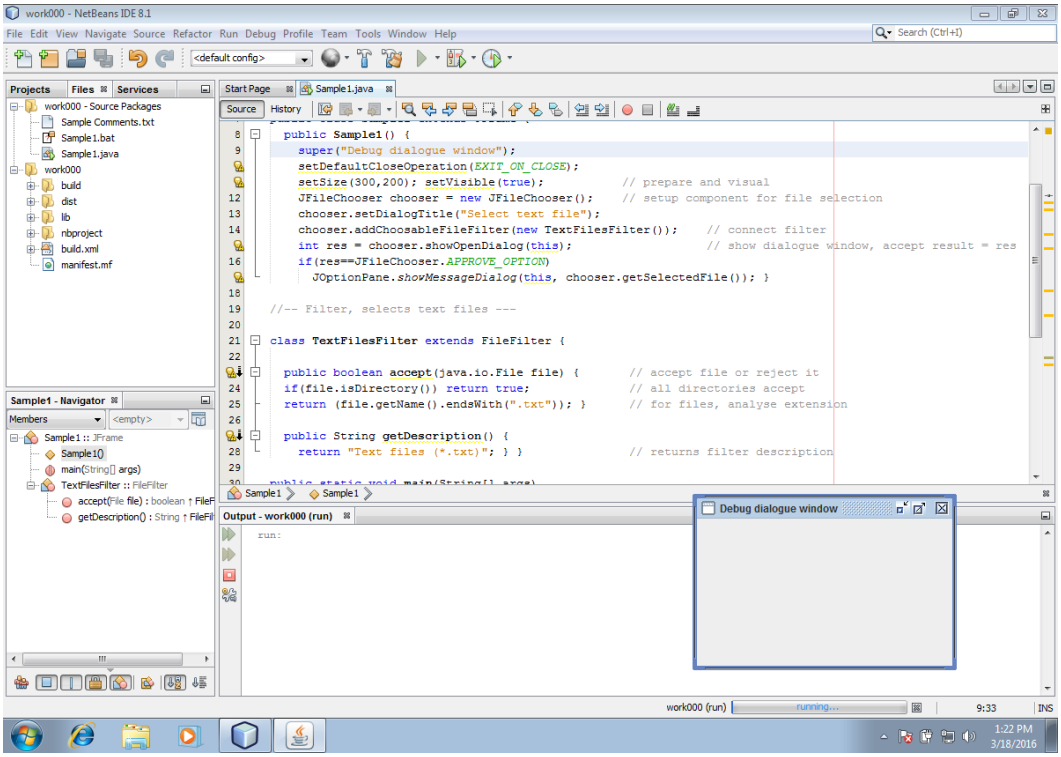


Рис.5 *Интегрированная среда разработки NetBeans, использует возможности Java Developer Kit ифункциональность Java Runtime Environment*

Критерий 4: удобство пользователя

Наверно, этот пункт должен быть первым, но поскольку мы обсуждаем *исследовательский проект*, он перечислен последним. Очевидно, EXE-файл приложения, запуск которого не требует подготовительных действий, удобнее для пользователя. В этом преимущество Free Pascal. Запуск Java-классов требует

инсталляции среды Java Runtime Environment (если она не является штатным компонентом ОС) и наличия пакетного BAT-файла для запуска JVM.

Набор Java-классов можно упаковать в выполняемый JAR-архив. Это не только уменьшит размер занимаемого дискового пространства, но и превратит приложение в *один* файл, запускаемый *«одним кликом»*, при условии, что среда JRE инсталлирована. Стремление к технической эстетике в сочетании с небольшой уловкой, позволит вам поместить в том же архиве библиотеки DLL и Kernel Mode драйвер для привилегированного доступа к системным ресурсам, если конечно эти компоненты нужны вашему приложению. Итак, достаточно *одного выполняемого архивного файла*.

Резюме

Ввиду сложности и многомерности исследуемой области, оставим вопрос приоткрытым, готовых рецептов выбора среды программирования нет по определению. Статья акцентирует внимание на аргументах для принятия решения, а решение, правильное в данном конкретном случае, должен принять разработчик или руководитель проекта.