Хорошилов Алексей Владимирович
Россия, Москва, Институт системного программирования РАН, khoroshilov@ispras.ru

Силаков Денис Владимирович
Россия, Москва, Институт системного программирования РАН, silakov@ispras.ru

МОБИЛЬНОСТЬ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ В СОВРЕМЕННЫХ УСЛОВИЯХ

Мобильность программного обеспечения обычно определяется как возможность переноса программного обеспечения (ПО) на широкий диапазон систем. Наиболее распространенными контекстами рассмотрения мобильности ПО являются следующие:

- переносимость ПО между различными аппаратными платформами (в первую очередь характеризуемыми архитектурой процессора);
- переносимость ПО между различными операционными системами (ОС);
- комбинация вышеперечисленных вариантов.

Реже мобильность ПО рассматривается в контексте переносимости между различными версиями одной операционной системы и в контексте переносимости между программными окружениями промежуточного уровня. В настоящей статье основное внимание будет уделено первым двум контекстам.

Техническая составляющая

Рассмотрим какие технические возможности в настоящее время существуют для обеспечения мобильности ПО. Начнем с проблемы переносимости приложения между различными аппаратными платформами с одной ОС.

Наиболее часто эта проблема решается на уровне исходных кодов приложения. Если приложение написано на интерпретируемых языках или языках программирования высокого уровня (Java, C#) приложение практически автоматически переносится на любую систему, где реализована соответствующая среда исполнения. При использовании промежуточного представления, такого как, Java байт-код, это можно делать даже без доступа к исходному коду. Если приложение написано на языке программирования более низкого уровня (Си, C++), то для обеспечения переносимости простой перекомпиляцией разработчик должен аккуратно учитывать возможные отличия между платформами, такие как, размеры базовых типов данных, их представление, неявные преобразования между типами и так далее. Это не является чрезмерно сложной задачей при достаточной культуре программирования, хотя и требует дополнительных усилий на проведение тестирования на целевой платформе.

В случае когда переносимость на уровне исходных кодов не доступна (например, при их отсутствии) возможно использовать переносимость на бинарном уровне, которая достигается при помощи эмуляции целевой аппаратной платформы. Эта эмуляция может быть реализована в аппаратуре (например, эмуляция x86 в Itanium IA-64), в операционной системе (например, динамический транслятор Rosetta, используемый в MacOS X, работающих на компьютерах Apple с процессорами Intel, для выполнения программ, предназначенных для машин с процессорами PowerPC [1]) или в виде независимого приложения (например, эмулятор Hercules для эмуляции мейнфреймов IBM S370, S390 и z-series).

Переносимость приложений между различными операционными системами обычно требует значительно больше усилий, чем переносимость между различными аппаратными платформами. В первую очередь, это связано с б**о**льшей разнородностью функциональности различных ОС, а также с существенно более широким и сложным интерфейсом их взаимодействия с приложениями.

Интерпретируемые языки и языки высокого уровня сами по себе уже не могут полностью решить эту проблему, так как большинству приложений не хватает базовых библиотек времени исполнения своего языка, а также им приходится взаимодействовать с множеством объектов, специфичных для различных ОС. Для языков низкого уровня ситуация еще более сложная ввиду ограниченности их библиотек времени исполнения. На помощь здесь приходит стандартизация

интерфейсов взаимодействия между приложениями и ОС (например, стандарт POSIX [2]), которая определяет контракт между приложениями и совместимыми ОС. К сожалению, в настоящее время стандарты такого рода поддерживаются далеко не всеми ОС, да и охватывают они лишь небольшую часть необходимой функциональности.

Для разрешения проблемы возможно использовать дополнительные кроссплатформенные библиотеки-медиаторы, которые реализуют требуемую функциональность на различных ОС и предоставляют приложению общий для всех ОС программный интерфейс. За последнее время разработано множество таких библиотек для различных видов функциональности: gtk+, Qt, wxWidgets, NSS, SDL, STL, OpenGL и многие другие. Конечно, библиотека должна удовлетворять требованиям приложения не только по предоставляемой функциональности, но и по многим других характеристикам: по лицензии, по качеству, по перспективам развития, по доступности исходного кода, по производительности и так далее. Если подходящей по всем условиям библиотеки не найдено, всегда можно разработать собственную библиотеку или доработать одну из существующих.

Альтернативным вариантом решения проблемы переносимости является использование эмуляторов и слоев совместимости исходной ОС на целевой ОС. В качестве примеров такого инструментария можно привести:

- wine — слой совместимости с WinAPI для Linux, MacOS, Solaris, FreeBSD и др. [3];
- cygwin — слой совместимости с Linux для семейства ОС Microsoft Windows [4].

Эти инструменты не предоставляют полный набор функциональности эмулируемой платформы, но в случаях, когда они покрывают все потребности приложения, следует рассмотреть возможность их использования для обеспечения переносимости уже разработанного ПО.

Достоинством wine является то, что он поддерживает переносимость приложений на бинарном уровне, без необходимости их перекомпиляции из исходного кода. Помимо wine эмуляция прикладного бинарного интерфейса ОС встречается в виде встроенного элемента в другие ОС. Например, FreeBSD поддерживает запуск исполнимых файлов Linux приложений, да и сам Linux имел поддержку запуска исполнимых файлов других UNIX-систем, хотя эта функциональность не оказалась востребованной.

Другие варианты бинарной переносимости включают в себя: стандартизацию бинарных интерфейсов и виртуализацию.

Стандартизация на бинарном уровне не является широко распространенным подходом. Не слишком удачная попытка создания такого стандарта была предпринята в 1990 году консорциумом Open Software Foundation (OSF) и называлась OSF/1. Более успешным оказался стандарт базового бинарного интерфейса различных дистрибутивов ОС Linux: Linux Standard Base [5-6].

Виртуализация позволяет запустить копию одной ОС внутри другой ОС, и, таким образом, любое приложение, предназначенное для первой ОС, может работать внутри второй ОС. Но следует учитывать, что этот способ достаточно накладен, как в смысле производительности, так и ввиду необходимости иметь лицензии на обе ОС.

Еще один подход к обеспечению мобильности ПО — это разработка Веб-приложений, в которых клиентская часть ПО взаимодействует с серверной посредством стандартов HTTP и HTML, что делает возможным работу с ПО на любой ОС, предоставляющей Веб-браузер.

Примеры коробочного ПО для массового рынка, использующего различные подходы к обеспечению мобильности приведены в таблице 1. Интересно отметить подход Google, который не побоялся положиться на слой эмуляции wine для запуска Google Picasa в ОС Linux, хотя до этого такой подход не имел столь широкого распространения среди разработчиков ПО. Подход с использованием библиотек-медиаторов более традиционен и применяется не один десяток лет.

Таблица 1

Примеры использования различных подходов обеспечения мобильности коробочного ПО

	Ориентация на стандарты	Медиаторы сторонних разработчиков	Собственные медиаторы	Эмуляция	Виртуализация
Примеры	*Ряд небольших POSIX приложений*	*Skype (библиотека Qt)*	*Mozilla Firefox (nss, nspr, ...)*	*Google Picasa (wine)*	---

Экономическая составляющая

Обеспечение мобильности ПО имеет как свои достоинства, так и свои недостатки. В зависимости от выбранного подхода эти недостатки могут включать:

- снижение производительности ПО;
- удорожание процесса разработки ПО;
- усложнение архитектуры ПО;
- дополнительные требования к квалификации разработчиков;
- осложнение использования возможностей, доступных только на некоторых платформах.

Это означает, что разработчикам и заказчикам ПО следует принимать решение о необходимости обеспечения переносимости в зависимости от условий конкретного проекта. Для краткосрочных проектов потребности в переносимости ПО, как правило, лежат на поверхности. Если же результаты проектов предполагается использовать в длительной перспективе, то анализ рисков должен быть более тщательным. В частности, нельзя оставить вне рассмотрения следующие риски:

- устаревание и выход из строя аппаратных и программных платформ, на которых базируется целевое ПО;
- зависимость от поставщиков базового и промежуточного ПО, требуемого как для работы целевого ПО, так и для его разработки;
- недоступность целевого ПО для части потенциальных потребителей.

Заказчики ПО при принятии решения о необходимости обеспечения переносимости ПО в конкретном проекте должны учитывать следующие факторы:

- собственный парк аппаратного и программного обеспечения, а также перспективы его развития;
- риски попадания в зависимость от поставщиков аппаратного и программного обеспечения;
- дополнительные затраты на обеспечение переносимости.

Разработчики заказного ПО должны нацеливаться на потребности своего клиента, но так же им следует учитывать и свои собственные потребности по возможному переиспользованию наработок данного проекта.

Например, в настоящее время при старте проектов по разработке Windows-приложений нельзя игнорировать вопрос работоспособности приложения на старой и новой версии ОС: Window XP и Windows Vista соответственно.

Разработчики коробочного ПО должны ориентироваться на результаты исследования рынка, чтобы иметь возможность корректно оценить окупаемость дополнительных затрат на обеспечение переносимости. При этом важно оценивать не только текущую ситуацию, но и прогнозировать перспективы ее развития. Например, разработчикам ПО, используемого в бюджетной сфере, следует серьезно рассмотреть вопрос о возможности работы приложения в ОС Linux, так как высшие представители Российского государства неоднократно выступали с недвусмысленными заявлениями о расширении использования этой ОС в бюджетных организациях.

Заключение

Вопрос мобильности ПО имеет длинную историю. За это время появилось множество подходов к его решению. Каждый подход имеет свои достоинства и недостатки и выбор наиболее подходящего варианта требует взвешенного подхода. В каких-то ситуациях, отказ от обеспечения мобильности ПО является правильным решением. В других ситуациях, такое

решение оказывается губительным для всего проекта. И наиболее обидно бывает, когда неправильное решение оказывается связано не с чьей-то ошибкой в анализе ситуации, а в банальной забывчивости и упущении проблемы, имеющей столь длинную историю. Поэтому главное, на что хотелось бы обратить внимание читателя, это то, что вопрос о необходимости и методах обеспечения переносимости ПО, должен ставиться и аккуратно рассматриваться в каждом проекте, причем в самом его начале.

Литература

[1] Apple Rosetta http://www.apple.com/rosetta/
[2] IEEE 1003.1-2004. Information Technology — Portable Operating System Interface (POSIX). New York: IEEE, 2004.
[3] Wine - Open Source implementation of the Windows API. http://www.winehq.org/
[4] Cygwin - Linux-like environment for Windows. http://www.cygwin.com/
[5] ISO/IEC 23360-1-8:2005, Linux Standard Base (LSB) Core Specification 3.1. Geneve: ISO, 2005.
[6] Linux Standard Base (LSB) Specification 3.2. http://www.linux-foundation.org/en/Specification