Руслан Богатырев

Гадание на кофейной гуще

Источник: Мир ПК, #02/1998

В прошлый раз мы с вами прокатились на колесе обозрения и познакомились с проблемами языков, связанных с Web-средой, и прежде всего с языком XML. Сегодня нас ждет экскурс в технологию Java.

Что и говорить, зима в парке аттракционов, а тем более у нас, где дыхание севера очень заметно, — далеко не самое веселое время года. Жизнь здесь замирает. И хотя взамен шумных каруселей среди искрящегося снега появляются другие забавы и развлечения, им непросто заменить своих летних собратьев.

Зима властно сковывает любой мало-мальский проблеск жизни. Наше колесо обозрения тоже не избежало печальной участи: сиротливо замерла его безмолвная громадина на фоне затянутого белой пеленой сумрачного зимнего неба. Но есть в этом пейзаже нечто сказочное, влекущее своим величественным спокойствием...

Внезапно что-то обрывает этот безмятежный сон наяву. Сильный пронизывающий ветер и крепкий мороз делают свое дело: хочется оказаться в тепле, поскорее согреться и уже оттуда полюбоваться сказочным видом. Хорошо, что неподалеку есть небольшой павильон — то ли кафе, то ли маленький ресторанчик, где можно выпить чашечку-другую горячего кофе. Туда мы и направимся.

Принять заказ подошла симпатичная девушка, но когда я попросил ее принести чашечку кофе, да притом яванского, она несказанно удивилась: подобные причуды нечастых посетителей, видно, были ей в диковинку. "Есть просто кофе — с сахаром или без", — не стараясь скрыть свое раздражение, пояснила она. "Хорошо, принесите, но только без сахара". После нескольких глотков стало ясно, что единственное достоинство поданного кофе — то, что он горячий. Крепость, вкус и аромат, по правде говоря, оказались очень посредственными. Изысканность этого напитка, который в XVIII-XIX вв. во Франции и России был неотъемлемым атрибутом роскоши и достатка, постепенно утрачена, подзабыта и технология его приготовления.

Но оставим на время разговор о кофе и обратимся к главной теме зимнего обозрения — столь модной и популярной ныне технологии Java.

Обвинения Sun

Центральным событием осени стал, безусловно, конфликт между Sun Microsystems и Microsoft — двумя великанами, которые, судя по названиям, хотят казаться этакими лилипутами. Этот конфликт вылился не только в череду исков и взаимных публичных обвинений — к выяснению отношений были подключены немалые силы не подозревавшей ни о каком подвохе общественности. Отбросим бессмысленные упреки в приверженности той или другой стороне (абсолютно непонятно, почему нужно обязательно занимать либо сторону Sun, либо сторону Microsoft, ведь обе они заслуживают жесточайшей критики) и спокойно во всем разберемся.

Итак, 7 октября 1997 г. Sun подала исковое заявление в окружной суд г. Сан-Хосе (шт. Калифорния), обвинив Microsoft в нарушении условий лицензионного соглашения о Java, заключенного в 1996 г. сроком на пять лет. Microsoft обвинялась в незаконном использовании товарного знака, заведомо ложной рекламе, недобросовестной конкуренции. Весь сыр-бор разгорелся из-за выхода в начале октября браузера Microsoft Internet Explorer 4.0 (MSIE 4.0), в котором реализация Java была выполнена, с точки зрения Sun, чересчур уж "творчески". Microsoft добавила ряд методов и полей классов Java, что лишало Java-программы переносимости на другие платформы и привязывало их к Windows, и при этом сохранила на MSIE 4.0 логотип Java Compatible.

В последовавших после этого официальных выступлениях первые лица компаний (президент Sun Скотт Макнили и президент Microsoft Билл Гейтс) четко обозначили свои позиции. Макнили уверял всех в том, что, предлагая на рынок свою трактовку, Microsoft по сути дискредитирует саму идеологию Java, а Гейтс удивленно разводил руками и говорил, что их реализация на сегодняшний день самая качественная и что она совместима со спецификациями Sun. Через неделю Sun выдвинула новые претензии к Microsoft, теперь уже на осязаемую сумму в 35 млн. долл. Sun

посчитала необходимым выложить сразу все козыри и обвинила Microsoft в намеренном распространении при выпуске SDK for Java 2.0 своих исходных текстов, относящихся к Java-среде. Судебное разбирательство обещает быть довольно продолжительным, тем более что текст лицензионного соглашения составлен весьма двусмысленно. Наверняка дополнительную остроту всему этому придадут взаимные встречные иски.

Нам же будет интересно разобраться в сути технологии Java, из-за которой и начались нешуточные баталии. Когда предъявляются обвинения в несоответствии, всегда возникает вполне законный вопрос: что является эталоном, с которым нужно проводить сравнение? В случае Java такой эталон вроде бы есть — это спецификации Sun, регламентирующие язык Java, виртуальную машину Java и необходимый набор классов. И пусть пока все это не утверждено в качестве международного стандарта, спецификации существуют и сравнение с ними возможно, тем более что Sun реализовала специальный эталонный набор Java Compatibility Kit (JCK). Sun сейчас предпринимает немалые усилия по формированию ISO-стандарта для Java. В середине ноября она получила благословение технического комитета JTC1 на статус PAS-координатора, который дает право представлять проекты стандарта на утверждение. Вся закавыка лишь в том, что развитие Java идет слишком уж стремительно и положения, закрепленные на бумаге больше года назад, сегодня уже вступают в противоречие с реальной ситуацией. В лицензионном соглашении между Sun и Microsoft четко указан эталон соответствия — Java Application Environment 1.0 (интересно, а знает ли кто-нибудь из специалистов, что представляет собой JAE 1.0 и как эти спецификации соотносятся с нынешним многообразием средств, включенных недавно Sun в свой последний инструментальный набор JDK 1.1.4?). И RMI (Remote Method Invocation), и JNI (Java Native Interface), и JFC (Java Foundation Classes) — все эти механизмы были специфицированы компанией Sun заведомо после 11 марта 1996 г., даты заключения пресловутого соглашения.

Так что с формальной точки зрения претензии Sun, по всей видимости, безосновательны. Позвольте, может возразить мне читатель, но ведь по сути Microsoft нарушает сам дух Java — принцип абсолютной переносимости ПО, а это куда важнее. Согласен, этот принцип действительно важен, и именно его Sun начертала на своем знамени, начиная в 1995 г. активную рекламную кампанию по продвижению Java, которая не завершилась и по сей день. Более того, она закрепила этот принцип в виде товарного знака Write Once Run Anywhere Safely (WORAS, "напиши однажды — запускай повсюду без опаски"). Однако благие намерения далеко не всегда соответствуют суровой действительности. Позже я покажу, как это выглядит в случае Sun, а пока лишь отмечу, что в пункте 2.8 все того же соглашения предусматривается право Microsoft расширять свои продукты на основе Java за счет дополнительных программных пакетов, так называемых VAOP (Value Added Open Package), причем они могут быть как переносимыми, так и непереносимыми. Следовательно, принцип WORAS может и не соблюдаться. Да и самого понятия WORAS в тексте соглашения нет.

Что такое Java?

Так что же такое Java — язык или платформа? Этим вопросом постоянно задаются в статьях, книгах и дискуссиях, посвященных Java. Я бы ответил на него так: Java — это язык и две платформы: операционная и инструментальная. Но прежде всего это технология, если, конечно, вкладывать в данное слово его изначальный смысл, т. е. понимать технологию как совокупность знаний о способах и средствах проведения производственных и информационных процессов.

Главная идея, которая положена в основу Java, — обеспечить независимость ПО от аппаратной и операционной платформ. Другими словами, система, созданная на Java, должна в идеале без какого бы то ни было вторжения в конфигурационные и исходные файлы функционально идентично исполняться на инородных платформах. При этом ставший уже привычным термин "переносимость", или "мобильность", с учетом столь жестких требований обретает более четкие формы. Более того, вновь берется на вооружение известный с незапамятных времен принцип кросс-разработки, когда инструментальная платформа (где создается программа) и целевая операционная платформа (где она исполняется) не совпадают.

Следующим важным положением технологии Java является принцип расширяемости систем, который реализуется за счет динамического связывания методов (динамической подгрузки) и за счет миграции кода в распределенной компьютерной среде, что особенно важно для Internet. Исполняемый код должен при необходимости передаваться по каналам связи на целевой компьютер и тут же включаться в работу.

Третье положение напрямую связано с предыдущим. Поскольку каналы связи подвержены помехам и несанкционированному вторжению, требуется обеспечить надлежащий уровень безопасности.

Все остальное — язык Java, виртуальная машина Java, многочисленные библиотеки классов — не является самоцелью, а призвано решать поставленные выше задачи. На весьма болезненном для Java вопросе быстродействия мы остановимся позже, а пока, забегая вперед и упреждая интерес нетерпеливого читателя, скажу, что бÿольшая часть дальнейших рассуждений с приведением технических деталей призвана лишь показать, как сама по себе хорошая идея в значительной степени дискредитируется в Java довольно неудачными решениями. На мой взгляд, они обусловлены в первую очередь тем ажиотажем, который был искусственно создан вокруг Java владельцами технологии. В результате не только возникает опасность профанации серьезных идей, но и сама компания Sun может пасть жертвой своей политики "открытой разработки" Java.

Недавно в журнале IEEE Micro была опубликована статья Р. Уркхарта "Открытое будущее Java" (см. перевод этой статьи в [1]). Говоря о достоинствах Java, автор отмечает два момента: "Во-первых, развитие и разработка этой платформы были организованы как открытый процесс, в котором могли принять участие все желающие. Во-вторых, Sun открыла рынок Java для конкуренции, объявив о том, что на использование и реализацию спецификаций этой платформы не будет наложено никаких ограничений".

Что касается первого, то любому, даже малосведущему человеку очевидно, что подобная открытость сродни открытости участия в тотализаторе на ипподроме (кстати, а не хотите ли вы абсолютно бескорыстно помочь какой-нибудь другой фирме в совершенствовании ее продукта?). Попытка автора объявить достоинством отсутствие отчислений за использование построенных с помощью инструментария Java целевых программ вызывает только улыбку: идея технологии никогда не нашла бы такой поддержки, если бы вам пришлось платить за это деньги.

Что же касается второго, то все мы на примере судебного иска Sun к Microsoft стали свидетелями совершенно обратного. Где-нибудь у себя, в укромном уголке, вы можете творчески развивать идеи Java, но если захотите показаться с ними в приличном обществе, будьте любезны соблюдать требования Sun. Механизм лицензирования вовсе не отошел в прошлое, а наоборот, весьма активно используется компанией в отношении всего, что связано с Java. Это справедливо не только для Java Platform (к осени 1997 г. у Sun уже было 117 фирм-лицензиатов), но и для ее многочисленных специфических модификаций (PersonalJava, EmbeddedJava и др.).

Так что всерьез имеет смысл говорить только о технологических новациях. Однако перед этим, если позволите, мне хотелось бы вернуться к своей уже основательно подостывшей чашечке кофе, а заодно возобновить нашу прерванную светскую беседу.

До середины XVII в. кофе был неизвестен европейцам. В Европу его привезли венецианцы. На протяжении XVIII в. как во Франции, так и в России это был модный напиток, распространенный среди привыкших к роскоши франтов и вельмож. Судя по гастрономическим запискам Александра Дюма-отца (который был знаменитым гурманом), кофе вошел в широкое употребление во Франции в 1808 г., в период континентальной блокады, которая лишила Францию сразу и кофе, и сахара. Чтобы сделать доступным резко подорожавший кофе и улучшить вкус напитка, казавшегося слишком горьким без сахара, повара стали добавлять в кофе истолченный цикорий. Именно в таком виде ставший относительно недорогим напиток приобрел популярность во всей Европе. В России в XIX в. более всего почитались ливанский и моккский (из Аравии) сорта кофе, а также кофе из Леванта; далее шли "островные" — бурбонский, куба, мартиника и, конечно же, ява.

Технологические новации Java

Говоря о новациях Java, следует выделить три составляющих: язык, систему поддержки и операционную платформу.

Система поддержки (RTS, run-time system), которая требуется любому языку, в случае Java сосредоточена в виртуальной машине. Ее основная задача — обеспечивать интерпретацию специального байт-кода, порожденного соответствующим компилятором. Принцип байт-кода и механизм работы виртуальной машины Java были позаимствованы из разработки Pascal-P двадцатипятилетней давности [2, 11].

Что касается операционной платформы (включающей RTL, run-time library), то она по-своему уникальна [2]. Помимо базовых классов — унификации средств современных ОС (можно выделить AWT, Abstract Windowing Toolkit), были построены собственные элементы операционной платформы (Java OS, JFC, RMI, JNI, JDBC, JMAPI, JavaBeans, JavaSpaces и др.). Разработан ряд специализированных операционных платформ: EmbeddedJava, PersonalJava.

Но ядром технологии все же является язык. Часто в отношении Java можно услышать термин "машинно-независимый", причем нередко также подчеркивается, что этим качеством обладает только Java. Позвольте, а что, Пролог — машинно-зависимый язык, или Смолток относится к этой категории, или, быть может, тот же Фортран? Неужто в мире программирования нет ничего, кроме Си, Си++ и Бейсика (точнее даже, его диалекта Visual Basic), которые подобным качеством похвастать не могут?

Когда речь заходит о языке Java [3], то среди новаций называют в первую очередь отказ от директив препроцессора, от оператора goto, от неявного преобразования типов, от указателей. Что ж, это можно только приветствовать. Правда, подобные новшества являются таковыми лишь для громадной армии программистов, пишущих на Си и Си++. То же самое касается введения проверки на пустой указатель и контроля выхода индекса за границы массива [4]. Все эти механизмы (как и другие средства обеспечения безопасности и надежности) давно являются штатными в других языках (в частности, в Аде, Модуле-2, Обероне).

Как известно, независимая компиляция (характерная для языков Си-семейства) исходит из того, что отдельно взятые единицы компиляции (файлы) не зависят от других и что все вопросы согласования взаимодействия откладываются до этапов компоновки и выполнения программного кода (часть из них так и остается неразрешенной). В случае раздельной компиляции наличие исчерпывающей информации обо всех используемых внешних элементах обязательно еще на этапе компиляции. Введение пакетов сразу ставит Java в один ряд с языками модульного программирования, которые как минимум два десятка лет используют концепцию модуля для разграничения интерфейса и реализации, для проведения раздельной, а не независимой компиляции и для эффективного контроля разработки крупных и серьезных проектов.

Немало "нового" было позаимствовано из неизвестных широкой общественности языков. Принцип интерфейса классов взят из языка Objective C, разработанного компанией NeXT. Механизм обработки исключительных ситуаций почти полностью позаимствован из Модулы-3 [5]. Справедливости ради следует отметить, что модель типов в Java по-своему уникальна — в ней происходит отказ от записей (структур) в пользу классов (в Обероне сделано с точностью до наоборот). В то же время принцип одинарного наследования, взятый на вооружение языком Java вместо более известного по Си++ множественного наследования, также не нов: он изначально был положен в основу Смолтока, Модулы-3 и Оберона. Средства многопоточного программирования построены на основе Модулы-3 и Ады (хотя идеологи Java подчеркивают здесь связь с языками Mesa и Cedar). В то же время принятая в Java схема чересчур ограничена для реализации систем реального времени и прежде всего для встроенных систем.

Что касается сборки мусора, то принципы внутренней реализации позаимствованы прежде всего из Лиспа (ведущий инженер Sun Гай Стил является автором двух диалектов Лиспа — CommonLisp и Scheme). Существует весьма устойчивый миф о том, что сборка мусора невозможна в Си++ и после "экзотических" для большинства программистов Лиспа и Смолтока появилась только в Java. Опровергнуть его несложно. Достаточно в качестве примера привести хотя бы работу 1989 г. Джоэла Бартлетта из лаборатории Western Research Laboratory, входящей в состав исследовательских подразделений компании Digital Equipment [6]. Но самое занимательное то, что часть "нормальных" языков, таких как Эйфель, Модула-3, Оберон, изначально располагала этим механизмом. Причем годы их появления (1986, 1988 и 1988-й соответственно) и авторитетность создателей гарантируют, что разработчики в исследовательских лабораториях Sun не могли об этом не знать.

Все изменения, которые были внесены в Java, выглядят необычными прежде всего для тех, кто привык иметь дело только с Си и Си++. А потому неудивительно, что, делая ставку именно на эту категорию программистов, авторы Java намеренно сохранили в синтаксисе языка черты Си.

История создания Java

Стоит ли в который уж раз возвращаться к истории Java, тем более что о ней и так написано немало статей? Да и книги о Java (только на русском языке я насчитал их более двух десятков), как правило, не обходятся без соответствующей главы. Думаю, что все-таки стоит. Именно в этом вопросе нужно быть крайне точным.

Открыв одну из отечественных книг, посвященных интересующей нас теме, я обнаружил такие слова: "Java задумывался как машинно-независимый и объектно-ориентированный язык программирования для Internet". Не хотелось бы разочаровывать автора, который, по-видимому, просто пал жертвой многочисленных слухов и выдумок, но это утверждение далеко от истины.

История Java окутана ореолом загадочности и таинственности. Поэтому немудрено, что с этой технологией связано бесчисленное множество мифов. Печально лишь, что мифы эти не просто до неузнаваемости искажают действительность — по большей части их авторами явились сами разработчики и владельцы технологии. Джеймс Гослинг мог снять многие вопросы еще в своем каноническом описании Java [3]. Однако в этой книге нет ни одной (!) ссылки на те языки, из которых были позаимствованы идеи Java. В конце приводится лишь никак не связанный с текстом список литературы для дополнительного чтения, где и упоминаются языки-конкуренты.

До января 1995 г. язык Java носил более прозаичное название — Oak ("дуб"). Автором и самого языка, и первоначального имени был Джеймс Гослинг, который в июне 1991 г. начал его разработку в исследовательских лабораториях фирмы Sun в рамках проекта Green (бывший Stealth Project). Oak изначально задумывался Гослингом как интерпретируемый диалект языка Си++, предназначенный для программирования миниатюрного устройства (наподобие карманного ПК), которое впоследствии получило название *7 (Star7). "Мне пришлось приложить максимум усилий, чтобы упростить интерпретацию и предусмотреть средства верификации байт-кода до его компиляции в машинный код", — писал впоследствии Гослинг. Таким образом, язык Oak был ориентирован на разработку операционной системы (Green OS) с прицелом на программирование бытовой электроники.

Как гласит история [7], когда в Sun Labs был закончен макет *7, президент Sun Скотт Макнили воскликнул: "Эта штука — прорыв. Теперь не подведите меня... Вы сделаете — мы победим. Мы продадим это. Мы убьем HP, IBM, Microsoft и Apple одним ударом" [8]. Шел 1992 год. Для доведения технологии до товарного вида была основана фирма FirstPerson, которая после непродолжительных мытарств с 3DO и Silicon Graphics бесславно закончила свое существование.

Нужно было что-то делать, и тут в июне 1994 г. вице-президенту Sun Биллу Джою пришла в голову (а вот как это произошло, история стыдливо умалчивает) идея переориентировать Oak на иные задачи — создание компактной ОС. В пожарном порядке был сверстан проект, который получил название LiveOak. Но уже в июле Патрик Нотон, автор графического интерфейса для *7, решил резко сузить фронт работ и сконцентрироваться только на Internet. В середине сентября 1994 г. Нотон вместе с Джонатаном Пейном приступил к разработке браузера WebRunner. Осенью 1994 г. Артур ван Хофф переписал компилятор Oak на самом Oak (до этого Гослинг реализовал его на Си). Так что худо-бедно к началу 1995 г. команда (правда, уже без ушедшего Нотона) располагала браузером и компилятором. Как же появилось название Java?

История его выбора довольно занимательна, хотя о ней почти никогда не пишут. При выпуске языка на рынок очевидно встал вопрос о новом названии: оно должно было быть ярким и запоминающимся и в то же время не конфликтовать с уже используемыми именами в товарных знаках других компаний. От старого названия Oak пришлось отказаться прежде всего из-за того, что схожее имя было у известной фирмы Oak Technologies. Обсуждение носило весьма бурный характер, и участники его так и не смогли в точности вспомнить, кто же предложил слово Java. Правда, Ким Полиз утверждает, что название придумала именно она. Что ж, в Sun Labs работали настоящие джентльмены: из вежливости они решили с леди не спорить (кстати, Ким ныне занимает пост исполнительного директора фирмы Marimba). Быть может, скрывая истинного героя, они хотели подчеркнуть силу коллективного творчества. Но все же Артур ван Хофф (ныне директор по технологиям в той же Marimba) проговорился: он убежден, что первым предложил это слово Крис Уорт, один из инженеров проекта LiveOak.

При выборе имени было несколько вариантов: Java, Silk, DNA, Pepper, Lyric, NetProse, Neon, WRL, Greentalk, Ruby, WebDancer, WebSpinner. В конце концов Ким Полиз предложила Эрику Шмидту, который занимал тогда пост директора по технологиям компании Sun, двух кандидатов: Java и Silk. При этом она настаивала на Java. Шмидт согласился. Таким образом, именно он стал крестным отцом Java (ныне Шмидт возглавляет компанию Novell). Логотип для Java (чашечка дымящегося кофе) был разработан Марком Андерсеном, тем самым, который создал запоминающиеся логотипы для Sun и Macintosh. Браузер WebRunner был переименован в HotJava. Наконец, 23 мая 1995 г. на выставке SunWorld '95 было официально объявлено о рождении нового языка.

А был ли у Java прообраз?

Не спешите возмущаться самой постановкой такого вопроса. Я задаю его не из праздного любопытства, просто в этом-то и состоит одна из "величайших" тайн компании Sun, руководство которой прилюдно категорически все отрицало, называя среди очень далеких конкурентов Java языки ScriptX (Kaleida Labs), Lingo (Macromedia) и Telescript (General Magic). Но историю ведь не перепишешь, как бы кому этого ни хотелось.

Так сложились обстоятельства, что 29 сентября 1994 г., в тот день, когда после двух недель напряженной работы руководству Sun был впервые продемонстрирован действующий макет браузера WebRunner, я находился в маленьком уютном городке Ульм на юге Германии. Там проходила международная конференция по модульным языкам (Joint Modular Language Conference), на которой были представлены Модула-2, Ада, Модула-3, Эйфель, БЕТА, Смолток. И все же в центре внимания оказались многочисленные работы по Оберону (объектно-ориентированному потомку Модулы-2) и операционной системе Oberon. То, что я тогда увидел, произвело на меня большое впечатление.

Помимо обширного инструментария, среди которого выделялся коммерческий 64-разрядный компилятор языка Оберон-2 для DEC Alpha/AXP, было немало законченных прикладных систем: геоинформационная система по Швейцарии, система математических вычислений Maple, система для фотосервиса Digital Kodak. Особняком стояли многочисленные коммуникационные компоненты системы Oberon: FTP, электронная почта, Telnet, Teletext/Telenews и WWW (!).

Проект Oberon стартовал в начале 1985 г. в Швейцарском Федеральном технологическом институте (ETH) в Цюрихе. Он велся под руководством профессора Никлауса Вирта (автора Паскаля и Модулы-2). За девять лет, к осени 1994 г., были созданы три языка (Оберон, Объектный Оберон и Оберон-2), ОС Oberon (перенесенная и на большинство известных тогда операционных систем за исключением разве что NeXTstep и Acorn RISC-OS), а также несколько видов рабочих станций (Ceres) со специализированным процессорным устройством, ориентированным на язык Оберон. Компилятор и ОС были столь компактными, что на Ceres их перекомпиляция занимала менее 15 секунд! Практически весь инструментарий, включая и некоторые компиляторы, свободно поставлялся со всеми исходными текстами и с подробной документацией. Желающие могли приобрести диск CD-ROM с инструментарием Oberon, уже выпущенный к тому времени в издательстве Addison-Wesley (!).

Но самая интересная новость была впереди. Оказывается, ученик Вирта по имени Михаэль Франц в феврале 1994 г. защитил в ETH диссертацию, которая называлась "Динамическая кодогенерация — ключ к переносимому программному обеспечению". В ней были четко обозначены проблемы виртуальных машин и предложен крайне простой и довольно необычный подход к достижению переносимости. Компилятор как бы разрезался на две части: "препроцессор" (front-end) и "постпроцессор" (back-end). Как известно, на первой фазе компиляции (анализе) обычно производится сканирование исходного текста и формирование синтаксических деревьев [9]. На второй фазе (синтезе) осуществляется оптимизация и генерирование объектного кода. Так вот, Франц предложил заканчивать компиляцию на первой фазе, а результат сохранять в виде особым образом (на основе семантического словаря) сжатых файлов. Вторую же часть компилятора он соединил с загрузчиком. Для Оберона загрузчик нужен все равно необычный, так как одной из особенностей ОС Oberon является динамическая подгрузка/выгрузка модулей.

В результате полуфабрикатный объектный код оказался очень компактным, и в совокупности с изначально быстрой генерацией кода для языка Оберон потери на загрузку программ-полуфабрикатов в смеси Ceres-Macintosh составили всего 10-20%. Эта технология получила название OMI (Oberon Module Interchange) и была встроена в ОС Oberon. В диссертации Франца не только подробно разбирался принцип формирования и хранения промежуточного представления, но и делался вывод о том, что для подобного класса языков он остается практически неизменным. Другими словами, для языка Java этот принцип вполне подходил. Кстати, понятие applet тоже фигурировало в диссертации Франца. После выпуска Java Франц к лету 1996 г. подготовил среду под названием Juice [10], выполненную в виде подключаемого модуля для браузеров Netscape и Microsoft. Модуль состоит из компактного варианта ОС Oberon и полнофункционального компилятора Оберона-2, размер которых (в Оберон-формате) в общей сложности составляет всего лишь 100 Кбайт.

Язык	Год выпуска	Автор(ы)	Где создан	Ключевые идеи
A. Универсальные прикладные языки				
Fortran	1954	John Backus	IBM	subroutine
Algol-60	1960	Peter Naur*	IFIP	if-then-else
Cobol	1960		CODASYL Committee	record
Basic	1963	John Kemeny*	Dartmouth College	routine
PL/I	1964	George Radin		fork, exception
Algol-68	1968	A.Wijngaarden*	IFIP	semaphore
B. Уникальные языки				
APL	1957	Kenneth Iverson	Harvard University	branch
Snobol	1962	Ralph Griswold	AT&T Bell Labs	string
Forth	1968	Charles Moore		stack/word
SETL	1969	Jack Schwartz	IBM	set/tuple/map
Icon	1974	Ralph Griswold	AT&T Bell Labs	generator
CLU	1974	Barbara Liskov	MIT	cluster
Postscript	1982	John Warnock*	Adobe Systems	page
Eiffel	1986	Bertrand Meyer	Interactive Software Eng.	assertion
Self	1987	David Ungar	Sun Labs	delegation
C. Языки Simula-семейства				
Simula	1962	Kristen Nygaard*		record class / coroutine
Smalltalk	1972	Alan Key*	Xerox PARC	object/class

BETA	1983	Kristen Nygaard*	Mjolner Informatics ApS	pattern
D. Языки Lisp-семейства				
Lisp	1958	John McCarthy	MIT	list
Planner	1967	Carl Hewitt	MIT	theorem
Scheme	1975	Guy Steele*	MIT	continuation
Common Lisp	1984	Guy Steele*	MIT	generic sequence
Haskell	1990	Paul Hudak*	University of Glasgow	functional array
E. Языки Prolog-семейства				
Prolog	1971	Alain Colmerauer*	Univ. of Aix-Marseille	unification
Parlog	1983	K.Clark*	Imperial College	AND-parallelism
CLP(R)	1986	Joxan Jaffar*	IBM Research	constraint
F. Языки C-семейства				
C	1972	Dennis Ritchie*	AT&T Bell Labs	address ariphmetic
C++	1986	Bjarne Stroustrup	AT&T Bell Labs	class
Objective C	1986	Brad Cox	Productivity Products	interface
Java	1995	James Gosling*	Sun Labs	package
Limbo	1996	Dennis Ritchie*	Bell Labs (Lucent Tech.)	implementation part
G. Языки Pascal-семейства				
Pascal	1970	Niklaus Wirth	ETH Zurich	name equivalence
Modula-2	1978	Niklaus Wirth	ETH Zurich	module
Oberon	1988	Niklaus Wirth	ETH Zurich	type extension

Oberon-2	1991	Hans Moessenboeck*	ETH Zurich	type-bound procedure
Component Pascal	1997	Cuno Pfister*	Oberon microsystems	component
H. Языки Ada-семейства				
Euclid	1976	Butler Lampson*	Xerox PARC	collection
Mesa	1976	J.Mitchell*	Xerox PARC	module
Ada	1977	Jean Ichbiah*	CII Honeywell	package/rendezvous
Cedar	1983	Butler Lampson*	Xerox PARC	thread
Modula-3	1988	Luca Cardelli*	DEC SRC, Olivetti Research	safe module
I. Языки параллельного программирования				
Concurrent Pascal	1972	Per Brinch Hansen	USC	par begin
Modula	1977	Niklaus Wirth	ETH Zurich	process
CSP	1978	Charles Hoare	Oxford University	cobegin
Edison	1980	Per Brinch Hansen	USC	critical region
Occam	1982	David May*	Inmos	channel
Linda	1985	D.Gelernter*	Yale University	tuple space
Obliq	1993	Luca Cardelli	DEC SRC	network object
* означает, что авторов было несколько				
Выделены языки, в той или иной степени оказавшие влияние на Java.				
Сокращения				
MIT — Massachusetts Institute of Technology;				
ETH — Swiss Federal Institute of Technology;				
PARC — Xerox Palo Alto Research Center;				
SRC — DEC Systems Research Center				

Но и это еще не все. Оказывается, в марте 1994 г. Михаэль Франц сделал в Sun Labs несколько докладов, и к весне того же года Билл Джой уже имел на руках всю необходимую информацию,

включая и диссертацию Франца. Джой стал одним из первых обладателей лицензии на ETH Oberon [11]. Все это необходимо учитывать при взгляде на нынешние проблемы Java.

Независимость от платформы

Этот аспект является, пожалуй, ключевым в технологии Java. Правда, здесь обязательно следует уточнить, о какой платформе идет речь: об аппаратной, операционной или языковой. С точки зрения аппаратной платформы важным элементом является внутреннее представление базовых типов данных: это касается расхождений в длине машинного слова (8, 16, 32, 64 или 128 разрядов), порядка хранения байтов в слове (какой байт идет в начале: младший или старший), внутренней структуры представления чисел с плавающей запятой (как хранится мантисса). Существенна также организация работы с памятью: доступ к машинным регистрам, к схеме представления оперативной памяти. Унификация аппаратной платформы в Java возложена на байт-код.

Языковая платформа подразумевает реализацию конструкций данного языка с помощью RTS (как хранятся в памяти массивы и другие структуры данных, как выполняются встроенные операторы и операции, как осуществляется инициализация и утилизация памяти, как реализованы исключения и поддержка параллелизма и пр.). Все это находится в ведении виртуальной машины Java.

Операционная платформа определяет взаимодействие программ с окружающей средой (прежде всего с операционной системой); в значительной степени она концентрируется в библиотеках поддержки (RTL), в сервисных библиотеках модулей/ классов (графика, коммуникационные и другие функции) и во внешних компонентах и API-модулях.

Независимость Java от платформы носит условный характер, и даже машинная независимость байт-кода Java отнюдь не решает проблему. Машинная независимость (в изначальном смысле этого слова) на уровне языка может быть обеспечена только для прикладных языков, которым нет надобности оперировать низкоуровневыми концепциями. Языки же системного программирования вынуждены прибегать к подобным средствам, в противном случае чаще всего будет страдать эффективность. В языках, претендующих на роль системных, необходимо иметь четкое разграничение между высокоуровневыми (безопасными и переносимыми) и низкоуровневыми (опасными и непереносимыми) элементами. К сожалению, ни Си, ни Си++ такого четкого разграничения не имеют. Однако другие языки подобными средствами обладают (например, Модула-2 с псевдомодулем SYSTEM и Модула-3 с концепцией SAFE-модулей и нетрассируемыми ссылками). Это, конечно, не означает, что машинно-независимый уровень всегда гарантирует переносимость. В любом языке (и Java здесь не исключение) есть ниточки, которыми он связан с внешним миром. Достаточно, скажем, при записи информации на диск в Windows указать путь файла в нотации Unix, и, если в системе поддержки нет соответствующего адаптационного механизма, вся переносимость рухнет.

Скрытые проблемы с переносимостью Java заключаются еще и в том, что механизм поддержки многопоточного программирования встроен прямо в язык, а ведь данный аспект столь сильно зависит от аппаратной и операционной платформы, что Никлаус Вирт вынужден был еще в 1977 г. отказаться от этой затеи и выбросить из Модулы встроенный механизм процессов (аналогов потоков). После чего, сохранив крайне упрощенный принцип сопрограмм, в Модуле-2 он перенес центр тяжести на библиотечную поддержку параллелизма.

Язык Java ныне занимает довольно странное положение: он уже не системный, но еще и не прикладной. Быть может, в данном случае стоит говорить о его чисто связной и транспортной функции. Сама эволюция Java-платформы (в частности, создание Java OS) и абсолютизация возможностей языка настоятельно подталкивают к превращению Java в язык системного программирования. Но если оставаться на позициях лицемерия и, показывая на индюка, называть его павлином, то единственный способ сохранять "машинно-независимый" фасад Java — до бесконечности через черный ход внедрять в него все новые и новые классы, часть которых реализовывать на других языках. Не в этом ли состоит стратегический план Sun?

Условием переносимости в языке Java служит принцип тотальной унификации с запретом потенциально непереносимых механизмов (прежде всего указателей и адресной арифметики). Столь крайний подход приводит к тому, что в библиотеках приходится предусматривать реакцию на все случаи жизни. Иными словами, в технологии Java властвует весьма сомнительный принцип: "Все, что не разрешено законом, запрещено". При этом, что характерно, проблемы переносимости полностью не исчезают, а лишь уходят в подполье: отказаться от использования системно-зависимых методов (native methods) для решения серьезных практических задач, а не для написания простейших аплетов практически невозможно. Так что если вы хотите сохранить верность Java, но не удовлетворены функциональностью и эффективностью нынешнего "законодательства" по Java, то

вам придется либо смирить гордыню, либо начать давление на Sun с целью принятия соответствующих "поправок", либо, наконец, "уйти в леса".

Безопасность

Безопасность Java — один из наиболее острых вопросов. Размеры статьи, к сожалению, не позволяют подробно разобрать даже основные проблемы, но кое-что я все же отмечу. Для обеспечения контроля существуют две противоположных стратегии. Первая состоит в том, чтобы отсекать любую мало-мальски подозрительную операцию (принятый в Java-машине принцип "песочницы", sandboxing), вторая — в том, чтобы маркировать поступающий код (принцип доверия, trusting, активно поддерживаемый Microsoft). Естественно, возможны и различные гибридные варианты. Изначально сделав ставку на "песочницу", фирма Sun стала заложницей этой идеи (сейчас, вводя системные и прикладные защищенные домены и двигаясь в сторону маркировки кода, она пытается как-то выправить положение).

"Песочница" сильно ограничивает программистов, и в реальных проектах им приходится прибегать ко всевозможным уловкам (Java обмануть несложно). Ладно бы, если бы ограничения помогали обеспечить настоящую безопасность, но это утопия. И дело здесь не в единичных просчетах в реализации, которые то и дело всплывают на поверхность. Та модель безопасности, которую столь долго оттачивала Sun, как теперь убедительно доказано, даже теоретически несостоятельна.

На прошедшей в октябре 1997 г. в Балтиморе конференции National Information Systems Security в своем нашумевшем докладе доктор Марк Лэдью высказал идею о том, что модель безопасности в Java, построенная на верификации байт-кода, ненадежна в принципе, — утверждение настолько серьезное, что перед представлением доклада потребовалось даже провести дополнительную экспертизу с привлечением специалистов из Национального центра компьютерной безопасности США (National Computer Security Center) и из Национального института стандартов и технологии (National Institute of Standards and Technology).

Вот к каким выводам пришел Лэдью [12]. Один из важных источников проблем, по его мнению, состоит в том, что формат файла класса допускает вольности в отношении языка Java. Другими словами, можно построить такой байт-код, которому не будет соответствовать ни одна Java-программа. Более того, в отличие от языка Java, байт-код Java содержит ряд низкоуровневых инструкций, в частности goto, которые позволяют делать все что угодно.

Основной фундамент, на котором строятся все рассуждения Лэдью, составляют изначальные расхождения исходного текста на Java и байт-кода. С учетом принятой на вооружение архитектуры виртуальной машины на этом-то и можно играть. Лэдью не ограничился одними теоретическими выкладками — на его Web-узле представлены весьма занимательные примеры агрессивных аплетов Java.

Проблем, выявленных доктором Лэдью, можно было бы избежать, если бы разработчики Java так не держались за весьма утопическую схему безопасности и за саму идеологию виртуальной стековой машины.

Проблемы производительности

Производительность — это ахиллесова пята Java. В чем же причина? В том, что байт-код Java, полученный в результате компиляции, содержит низкоуровневый набор инструкций. И если осуществлять его интерпретацию, то разница в производительности по сравнению с лучшими компиляторами Си++ составит несколько порядков (!). Для борьбы с этим недугом придумали ряд приемов.

Один, более известный, носит название JIT-компиляции (just-in-time) и состоит в том, что байт-код (точнее, отдельные его фрагменты, отвечающие за конкретные методы) "на лету" ретранслируется в чистый (native) код данного процессора. JIT-компиляция должна быть быстрой, иначе она полностью дикредитирует идею. Осуществлять ее можно не только на клиентском компьютере, но и на сервере, однако тогда мы столкнемся с проблемами безопасности и объема кода, передаваемого по каналам связи. Как показали многочисленные эксперименты, проведенные различными исследователями, наиболее быстрым JIT-компилятором на сегодняшний день обладает извечный противник Sun —

корпорация Microsoft. Но и ее продукт в лучшем случае достигает лишь четверти возможной производительности.

В середине февраля 1997 г. компания JavaSoft, являющаяся отделением Sun, объявила о покупке небольшой фирмы LongView Technologies LLC, которая создала технологию HotSpot, способную, по мнению JavaSoft, радикально поднять производительность Java. Первоначально HotSpot разрабатывалась с ориентацией на Smalltalk (IBM VisualAge), теперь ей нашли и другое применение.

Технология HotSpot противопоставляется JIT-компиляции и положена в основу нового поколения платформы Java. Оно найдет свое отражение в инструментарии JDK 1.2, чей выпуск намечен на середину 1998 г. (JDK 2.0 — на конец 1998 г.). HotSpot проводит выборочную оптимизацию только тех участков кода, которые в ходе выполнения программы определяются как неоптимальные. По мнению создателей этой технологии, такой адаптивный подход может дать лучший эффект, чем статическая компиляция.

Развивается и направление flash-компиляции. Здесь следует сказать о работах компании Asymetrix, которая в дополнение к 32-разрядному компилятору Java для Windows 95/NT в начале 1998 г. планирует выпустить версию и для Solaris. Кстати, работы финансирует небезызвестный Пол Аллен, вместе с Биллом Гейтсом основавший Microsoft. Flash-компиляция основное внимание уделяет все же не качеству генерируемого кода, а скорости динамической компиляции (она, по словам разработчиков, в два-пять раз превышает привычные JIT-компиляторы).

Другой, менее известный подход — NET (native executable translation) — был разработан в университете штата Иллинойс [13]. Он базируется на идее статической, а не динамической ретрансляции байт-кода Java в чистый код целевого процессора. Одна из серьезных проблем — отображение стековой модели байт-кода на регистровую модель большинства современных процессоров, в которых к тому же используется более эффективная организация памяти. Чтобы у читателя сложилось более четкое представление о производительности различных компиляторов, приведу некоторые сравнительные результаты.

Итак, исследовательская группа из Иллинойской компьютерной лаборатории аэрокосмических систем при поддержке NASA, Sun и ряда других компаний провела эксперименты по сопоставлению характеристик компиляторов и интерпретаторов Java различной структуры (JIT, NET). Эксперименты проводились, в частности, с целью определения реальных параметров разработанного в университете штата Иллинойс компилятора Impact NET, построенного на основе NET.

Сравнению подвергался код тестовых примеров (Sieve, LinPack, SPEC95, Otello, Cup, Pi, javac 1.0.2), полученный с помощью Microsoft Visual C++ 4.2, Microsoft JIT 1.0, Sun JDK 1.0.2 (интерпретатор) и Impact NET. В результате максимальное быстродействие оказалось у Visual C++ (чего и следовало ожидать), а наихудшее показал интерпретатор Sun (что тоже понятно). Поэтому результаты остальных я приведу в процентах по отношению к эталону и в разах по отношению к интерпретатору. Быстродействие Microsoft JIT составило в среднем 23% от Visual C++ и в 5,6 раза превысило быстродействие JDK. Для Impact NET эти показатели составили 54% и 17,3 раза.

Почему даже при статической ретрансляции не удается приблизиться к характеристикам компиляторов Си++? Дело в том, что первичная трансляция в байт-код при переходе на более низкоуровневые команды не только разрушает исходную структуру программы, но и вносит, в частности, такие неприятные для генератора чистого кода моменты, как постоянные инструкции работы со стеком (pop, push). В работе, проведенной в 1997 г. в крупнейшем исследовательском центре IBM (IBM T.J. Watson Research Center) [14], было показано, каким образом можно попытаться справиться хотя бы с частью возникающих проблем.

В связи со всем вышесказанным встает вполне законный вопрос: а зачем нужно было создавать все те сложности, которые теперь героически преодолевают многие ученые и инженеры? Ведь подход Франца по сути решает эту проблему, поскольку в нем полуфабрикатный код является в принципе самим исходным текстом, но уже воспринятым компилятором. Таким образом, для проведения последующей оптимизации с привлечением лучших методик, отточенных на компиляторах Си++, есть вся необходимая информация, которой, увы, нет у байт-кода Java.

Для осуществления полноценной оптимизации, требующей информации не только о командах, но и о начальной структуре программы, необходимо определенное время, которого у динамических кодогенераторов (что в подходе JIT, что у Франца) просто нет. Выход Франц нашел в проведении двухфазной компиляции, когда в момент исполнения кода, полученного экспресс-способом, ведется фоновая оптимизация, требующая примерно в пять раз больше времени, а затем модули незаметно подменяются оптимизированными вариантами.

Тем временем Sun в буквальном смысле поймали на плутовстве. В середине ноября 1997 г. президент Sun Software Жанпитер Шердер был вынужден принести официальные извинения в связи с разразившимся скандалом относительно искаженных результатов тестирования производительности компилятора Java для Sun Solaris 2.6. За несколько дней до этого компания Pendragon Software обнаружила, что при обработке штатного тестового набора CaffeineMark 3.0 компилятор подставляет уже заранее подготовленный объектный код и тем самым существенно (в 300 раз!) искажает истинную производительность.

Обнаружить это удалось довольно просто. Специалисты Pendragon, сравнивая компилятор Sun с компиляторами других фирм, были удивлены его крайне высокой производительностью. Заподозрив неладное, они слегка модифицировали исходный текст на Java для тестового набора. И компилятор "обознался". Он не "осознал", что ему подсунули тот же тест и честно стал генерировать код по обычным правилам (а не подставлять уже "отточенный"). Падение производительности оказалось ошеломляющим. Представители Sun объяснили это случайным проникновением в финальную версию некоего отладочного промежуточного варианта, в котором якобы проводились эксперименты с оптимизацией по удалению недостижимого кода (dead code). Однако специалисты Pendragon внимательно все перепроверили и обнаружили в генераторе кода конкретный фрагмент, который и занимался "шулерством".

Возможные последствия "недоделок"

Недавно мне попалась на глаза одна статья, в которой автор безапелляционно заявлял, что ныне всем злопыхателям, писавшим разгромные статьи о грандиозном проекте Java, не осталось ничего другого, как замолчать. (Что-то мне таких статей ни в нашей, ни в зарубежной прессе не попадалось.) Помолчать, конечно, можно, да вот нетрудно сообразить, кому от этого будет польза: всем, кто делает большие и малые деньги на прибыльном бизнесе, связанном с Java. Приведу лишь один свежий пример. Серьезные изъяны в реализации технологии Java, выполненной фирмой Sun, естественно, только на руку всем тем компаниям, которые предлагают свои "лекарственные препараты", нейтрализующие эти изъяны. Одна малоизвестная американская фирма Finjan Software недавно выпустила на рынок пакет SurfinShield — средство обеспечения безопасности Java для Unix.

Следом за этим уже знакомый нам доктор Марк Лэдью решил проверить SurfinShield на устойчивость и обнаружил, что продукт никуда не годится. Об этом он открыто написал на своей Web-странице, где поместил подробные аргументы и тексты контрпримеров. Реакция не заставила себя долго ждать. В течение октября-ноября Finjan Software приняла все меры, чтобы заставить неугодного профессора замолчать (хорошо, что до наших способов разрешения конфликтов они пока еще не дошли).

Фирма написала письмо, полное надуманных обвинений, руководству Georgia Tech School of Mathematics — учебного заведения, в котором преподает Лэдью, — и в нем потребовала лишить Лэдью Web-страницы, что испуганное начальство тут же и поспешило сделать. Тогда Лэдью "попросил убежища" у другой фирмы (Reliable Software Technologies), которая ему не отказала. История борьбы с "умниками" вроде Лэдью наверняка еще не закончилась, но она наглядно демонстрирует, чем нередко сопровождается активное продвижение на рынок сырых и недоброкачественных программных продуктов.

Или совсем не пить кофе, или пить самый лучший

В книге об этикете, изданной в конце прошлого века в Петербурге, в отношении кофе высказана весьма интересная мысль: "Кофе имеет то общее с поэзией, что в нем, как и в поэзии, посредственность никуда не годится. Или совсем не пить кофе, или пить самый лучший".

Я все время задаю себе вопрос: зачем Sun понадобилось идти на заведомую авантюру? Если уж так нужна была "чистота идей", неужели трудно было внимательно разобраться с тем, что сделал ETH? Почему, прекрасно зная о существовании более разумных решений, Sun пошла своим, доморощенным путем (и потащила за собой огромное количество людей)? Частично ответ на эти вопросы в одном из интервью недавно дал Эрик Шмидт, объяснивший, что главной задачей тогда было добиться построения собственного архитектурного франчайзинга, альтернативного франчайзингу Microsoft. Иными словами, получить контроль над архитектурой, для которой будет написано огромное число приложений. Sun просто боялась, что ее опередят. Началась

широкомасштабная кампания "открытой разработки" Java. "История вас не забудет", "Все для фронта, все для победы" — вот какие девизы были начертаны на знамени Sun.

По мнению ряда аналитиков, в частности, из фирмы Forrester Research, если Sun не начнет откровенную монополизацию и диктат в области Java, то эту технологию неизбежно ждет появление нескольких ветвей развития, как это произошло с Unix. Интересно, удастся ли Sun вести свою "демократическую" линию так, чтобы ее не спутали с "тоталитарной диктатурой" Microsoft?

Идея инвариантности ПО сыграла огромную роль в деле "раскрутки" марки Java. О том, чего добились маркетинговые службы, паразитируя на этой идее, наиболее ярко сказал (уже будучи президентом Novell) сам Эрик Шмидт: "Когда вы произносите слово "Java", то, что бы ни скрывалось за ним, на первый план выступают позитивные аспекты фирменного названия. Вы чувствуете: это нечто солидное, люди возбуждаются, стремятся приобщиться. Даже таксисты знают: Java — это класс!"

Обратите внимание, постоянно подчеркивается тот факт, что та или иная система сделана на Java. Уже одно это должно заставить учащенно биться сердца восторженных потребителей.

Есть русская народная сказка о том, как солдат варил кашу из топора. Сюжет ее незатейлив, зато подтекст весьма глубок. Шел как-то старый солдат на побывку, заглянул в ближнюю избу и, проголодавшись с дороги, попросил у хозяйки перекусить. Однако старуха поскупилась: "Сама ничего не ела: у меня шаром покати". Тут солдат приметил под лавкой топор без топорища: "Коли нет ничего, можно и из топора кашу сварить". — "Как из топора?" — всплеснула руками хозяйка. Солдат попросил котел, вымыл топор, налил воды и поставил на огонь. Немного погодя достал ложку, помешивает варево. Попробовал. "Ну как?" — спрашивает старуха. "Скоро будет готова, жаль вот только, соли нет". — "Соль-то у меня есть, посоли". — "Сюда бы горсточку крупы", — говорит солдат. Старуха принесла из чулана крупу. "Эх, хороша каша, как бы чуточку масла, вовсе пальчики оближешь". Сдобрили кашу. "Бери ложку, хозяюшка". Стали есть кашу да нахваливать. "Вот уж не думала, что из топора этакую добрую кашу можно сварить", — дивится хозяйка. А солдат ест да посмеивается.

Будущее Java

Вопрос о будущем Java волнует очень многих [15]. На мой взгляд, во многом оно зависит от того, смогут ли специалисты преодолеть навязанные стереотипы и непредвзято взглянуть на эту технологию. Ведь то, что наработано за эти два с лишним года, содержит богатейший материал, который может быть переведен на куда более прочный фундамент. Я не имею в виду обязательно смену языка. Java при всех его недостатках язык неплохой. Свою побочную задачу — приобщение программистов, работающих на Си и Си++, к тем достижениям в области языков программирования, которыми они были обделены все эти годы, — он решает вполне добротно. Кроме того, что ни говори, Java обладает цельностью: он создавался одним человеком и потому не стал жертвой многих компромиссов, как правило неизбежных при работе согласительных комиссий.

Сосредоточившись на Internet и поставив во главу угла весьма спорную идею косметического улучшения графического интерфейса за счет аплетов (всевозможных "шпунтиков", "прибамбасов", "причиндалов"), Java все же многое потерял. В загоне оказались такие важнейшие области его применения, как научные расчеты и встроенные системы. То, что только в 1997 г. сделала фирма Visual Numerics, создав JNL (Java Numerical Library), можно назвать пока лишь первым робким шагом на пути к освоению этой области. А ведь унификация научных расчетов с ориентацией на параллельные архитектуры — крайне актуальная задача, решение которой с учетом ревизии богатой кладовой вычислительных алгоритмов, реализованных в свое время на Фортране, давно назрело, и инициатива OpenMP — яркое тому подтверждение.

Что касается встроенных систем, то заминки Sun, стремящейся объять необъятное, крайне негативно воспринимают прежде всего японские производители бытовой электроники. Они не хотят, чтобы их продолжали кормить обещаниями и (как в случае с EmbeddedJava) лишь обозначали технологии, не доводя их до полноценной реализации (первая реализация платформы Personal Java 1.0, предназначенная для рынка бытовой электроники, была представлена лицензиатам фирмы Sun только в начале января 1998 г.). Здесь уместно вспомнить историю Java и ту область, для которой собственно и создавался язык Oak. Пока Sun раскачивалась, некоторые фирмы из США, Европы и Японии уже подготовили свои решения. Среди них Inferno, Portos, JV-Lite и др. Не осталась в стороне и Microsoft, развернувшая проект Millennium и модернизацию Windows CE.

Самое главное в этой ситуации — критически оценивать реальные возможности Java. Однако этому мешает раздутая вокруг него шумиха. Приведу лишь одну цитату из статьи в специализированном журнале, ориентированном на профессиональных программистов и выпускаемом под эгидой IEEE Computer Society [16]:

Java — это настоящий язык-трудяга. Это не результат чьей-то диссертации, это язык для работы. Java покажется очень знакомым самым разным программистам, поскольку мы предпочитаем делать проверенные вещи. <...>

Итак, что же такое Java? Java ощущаешь как игривый и гибкий язык. Вы можете создавать с его помощью такие вещи, которые сами являются гибкими. Java ощущаешь как детерминированный язык. Если вам хочется, чтобы он что-то сделал, то просто попросите его об этом. В нем не видится ничего опасного: вы можете спокойно попробовать что-то сделать, и если окажетесь неправы, то быстро получите сообщение об ошибке. Java ощущаешь как очень богатый язык. Мы постарались снабдить его большой библиотекой классов. Поэтому не откладывайте дело в долгий ящик, а садитесь за компьютер и пишите свой код.

Не складывается ли впечатление, что перед вами выдержка из рекламного объявления? А ведь эти слова, опубликованные летом 1997 г., принадлежат Джеймсу Гослингу, не только автору Java, но и человеку, который защитил диссертацию в известном университете Карнеги-Меллон, связанную с проектом Andrew Windows System, и который ныне является вице-президентом компании Sun Microsystems.

"Представьте себе, — не в силах сдержать возмущение, комментирует приведенную цитату Вирт, — что эти слова были написаны в 60-е годы, и замените слово "Java" на слово "Алгол". Автора сочли бы человеком психически ненормальным, слова его большей частью чужды науке и не имеют с ней ничего общего. Сегодня никто даже не возмущается, нет никакой реакции от "научного" сообщества. Как же низко могла пасть "информатика"? И это делается с молчаливого одобрения такой уважаемой организации, как IEEE Computer Society?"

В заключение нашего зимнего обзора давайте немного расслабимся и погадаем. Для этого достаточно всего лишь раздобыть немного кофейной гущи (сорт кофе уже не имеет значения, так что гоняться за яванским необязательно). При этом надо, правда, помнить, что гадание на кофейной гуще самое надежное и опасное, потому что всегда сбывается.

Положите гущу в чашку, накройте блюдечком, а потом опрокиньте все это так, чтобы гуща пристала к стенкам чашки, после чего снимите блюдечко, налейте в него воды, возьмите перевернутую чашку за дно и трижды опустите ее в воду, произнося: "Верность, дружба и согласие".

Теперь посмотрите на стенки чашки. Силуэты животных предвещают опасность, огорчения. Очертания рельефов местности и растений — ссоры, неудачи. Силуэты людей резко усиливают нынешнее положение вещей. Здания — это счастливое предзнаменование, сулящее нежданное обогащение.

Что же видите вы?

Язык, как известно, во многом определяет мышление человека. Это справедливо и для языков программирования. Не секрет, что каждый из них задумывался для решения вполне определенных задач, но нередко судьба распоряжалась им совсем не так, как того хотел автор. Идея создания универсального языка, который великолепно (или хотя бы добротно) решал бы все проблемы, до сих пор будоражит умы. Возможно ли такое? Кто знает! Но, по правде говоря, в это верится с трудом. Полувековой опыт компьютерного программирования говорит скорее о невозможности построить универсальный язык. Более того, все отчетливее проявляется кризис программирования, разрешить который будет под силу новым языкам, не только обогатившимся новыми идеями, но и позаимствовавшим немало полезного у своих предшественников. В обширном семействе существующих языков (активно используемых или основательно подзабытых) есть такие, которые практически не нашли продолжения. Но, быть может, именно заложенные в них идеи окажутся благодатной почвой для языков будущего?

С появлением Java все остальные языки оказались фактически оттеснены на задний план. В наши дни новичкам на ниве программирования довольно трудно разобраться в том, какое место занимает Java среди себе подобных, — а ведь их сегодня насчитывается несколько сотен. К сожалению, построить генеалогическое древо, достаточно аккуратно отражающее преемственность между

языками, очень сложно. К тому же в связи с постоянной эволюцией языков и рождением многочисленных диалектов его структура заведомо будет чересчур запутанной.

Вот почему я решил прибегнуть к табличной форме представления, отобрав около 50 наиболее важных, с моей точки зрения, языков, которые в значительной степени определяли (или будут определять) дальнейшие пути развития программирования. Сюда не попали ни Web-языки, ни различные языки сценариев, ни специализированные языки (4GL), ни языки баз данных, ни особые диалекты, имеющие мало общего со своими предками.

Можно заметить, что языки обычно появляются на свет из университетов, исследовательских центров, мощных компаний. При этом удельный вес исследовательских центров очень высок. История показывает, что без поддержки промышленности языки широкого распространения не получают. Исключение составляют Бейсик, Лисп, Пролог и Паскаль. Но ни один из них (в классическом виде) не может быть отнесен к языкам системного программирования.

Время становления языка (от официального окончания разработки до начала активного использования) составляет в среднем 3-10 лет. Java представляет собой особый случай: его выход на рынок произошел менее чем через год.

Литература

1. Urquhart R. Открытое будущее Java // ComputerWeek-Moscow, 1997, #31, с. 40-41.
2. Богатырев Р. Феномен технологии Java // ComputerWeek-Moscow, 1996, #23, 31.
3. Арнольд К., Гослинг Дж. Язык программирования Java. СПб: Питер, 1997.
4. Хофф А. ван. Программирование на языке Java // ComputerWeek-Moscow, 1997, #27, с. 24-27.
5. Гослинг Дж. Что такое Java // ComputerWeek-Moscow, 1997, #29, с. 21-22.
6. Bartlett J. Mostly-Copying Garbage Collection Picks Up Generations and C++ // DEC WRL, 1989, Technical Note #12.
7. O'Connel M. Java: The inside story // SunWorld, 1995, #7.
8. Гагин А. История Java. // Планета Internet, 1996, #1, с. 28-38.
9. Kistler Th., Franz M. A Tree-Based Alternative to Java Byte-Codes // University of California at Irvine, Dept. of Information and Computer Science, 1996, Technical Report #96-58.
10. Богатырев Р. Java и Juice: дуэль технологий?! // Компьютерра, 1996, #34, с. 30-33.
11. Франц М. Java: критическая оценка // Мир ПК, 1997, #8, с. 56-60.
12. Ladue M. When Java Was One: Threats from Hostile Byte Code. 1996. <http://www.rstcorp.com/hostile-applets/ index.html>
13. Cheng-Hsueh A. Hsieh e. a. Optimizing NET Compilers for Improved Java Performance // IEEE Computer, 1997, v. 30, N 6, pp. 67-75.
14. Ebcioglu K., Altman E., Hokenek E. A Java ILP Machine Based on Fast Dynamic Compilation // IBM T.J. Watson Research Center, 1997.
15. Siddalingaiah M. The future of Java — rhetoric or reality? // JavaWorld, 1997, December.
16. Gosling J. The Feel of Java // IEEE Computer, 1997, v. 30, #6, pp. 53-57.