Перенос приложений и проблема 2000 года

Андрей Николаевич Терехов
Андрей Андреевич Терехов

В настоящее время повышенный интерес вызывает проблема переноса устаревших приложений (по-английски — legacy systems) на новые языки и аппаратные платформы. Это вызвано целым рядом причин. За последние десятилетия в сфере программирования произошло множество радикальных перемен. Развитие электроники привело к ошеломляющей миниатюризации компьютеров. Были изобретены новые технологии, такие как объектно-ориентированное программирование. Сменилась идеология общения с пользователем, появились современные графические интерфейсы. Были созданы и впоследствии всемирно признаны новые языки программирования, такие как C++, Java и Visual Basic. Наконец, широкое распространение получила сеть Internet и связанные с ней технологии.

В связи с этим возникла необходимость в модификации устаревших систем, написанных на таких языках, как COBOL, PL/I и т.д. для mainframe-компьютеров. Преобразования этих систем не сводятся только к трансляции со старых языков программирования в новые. Не последней среди причин для переноса является нашумевшая «проблема 2000 года». Суть проблемы заключается в том, что во многих используемых ныне системах для хранения года были зарезервированы только две цифры, подразумевая даты двадцатого века. 1 января 2000 года такие системы начнут выдавать ошибки. Решением этой проблемы занимается сейчас множество компаний. На данный момент рынок «проблемы 2000 года» оценивается в 600 миллиардов долларов.

Какие же существуют средства для борьбы со старением систем?

Задавшись этим вопросом, мы предприняли поиск по Internet, результатами которого и хотим поделиться с вами. Отметим, что основное внимание уделялось системам, написанным на языке COBOL.

«Не надо чинить то, что еще не сломалось»

Очень распространенный подход, основным достоинством которого является, конечно же, максимальная дешевизна. Однажды написанные и отлаженные программы поддерживаются небольшим коллективом программистов. В рамках одного и того же языка иногда возможна некоторая модернизация существующих программных комплексов, а иногда и перенос программ на другие платформы.

В последнее время появилось большое количество компиляторов, расширяющих язык COBOL в духе новых технологий. Например, MicroFocus предлагает Cobol Workbench (см. **http://www.microfocus.com/enterprise/wb40.htm**), поддерживающий объектно-ориентированное расширение COBOL Object Cobol. В комплекте с предлагаемым той же фирмой пакетом NetExpress (см. **http://www.microfocus.com/dcomp/netex.htm**), позволяющим создавать современный GUI-интерфейс для COBOL-приложений, это является мощным инструментальным средством разработки.

Своевременно звучат и предложения другой крупной компании, производящей компиляторы COBOL. На сайте Fujitsu Corp. (см. **http://www.adtools.com/products/netcobol.htm**) рекламируется NetCobol, который умеет генерировать из исходной программы байт-код языка Java. Этим достигается почти полная переносимость COBOL-программ. Кроме того, в стандарт COBOL добавлены выражения EXEC JAVA и EXEC HTML. В результате в плане возможности управления Web-страницами COBOL встал в один ряд с Java и C++.

Кстати, возможность генерации из COBOL байт-кода языка Java является основой для трансляции из COBOL в байт-код, а затем декомпиляции с получением в результате программы на Java. К этому методу мы еще вернемся.

И все же основным недостатком COBOL остается... сам COBOL. Не имеет смысла приукрашивать и лакировать COBOL — он не станет от этого лучше. Невозможно косметическими средствами «вылечить» язык, в котором нет даже понятия процедуры.

«А давайте вместо этого купим Microsoft Word...»

Другим возможным сценарием является такой: неожиданно для себя вы выяснили, что в современном окружении существует продукт, исполняющий в точности те же функции, что и ваша система, например, автоматизирует бухгалтерию. Тогда вместо переписывания старой системы можно купить новый продукт,

потратив некоторое время на его настройку на ваш конкретный случай. Если на вашем предприятии дела обстоят именно так, то считайте, что вам невероятно повезло, и незамедлительно приступайте к реализации этого плана!

Единственная загвоздка состоит в том, что таких случаев — менее 5%. В большинстве случаев устаревшие системы накапливали опыт, специфичный для конкретного предприятия. Именно поэтому тяжело заменить такие приложения продуктами «со стороны».

«Нам очень нужны эти знания»

Так что же делать, если вы отчетливо осознаете необходимость замены вашей системы и в то же время не можете себе позволить потерять накопленные за долгие годы знания? В таком случае единственным возможным решением является перенос, или **реинжиниринг**.

Каким бывает реинжиниринг?

Основным (и до недавних пор единственным) вариантом традиционно считается «ручной» реинжиниринг, когда для переделки большой системы нанимается соответственно большой штат программистов, каждый из которых отвечает за свой кусок программы. Особенно часто это практикуется в Америке, где набирают большой штат программистов сравнительно невысокой квалификации, но аккуратных и усидчивых. Обычно таких программистов массово нанимают в Индии, благо там рабочая сила на порядок дешевле. Такие фирмы имеют общее название — reengineering factories.

Недостатки такого подхода очевидны. Во-первых, низкое качество перевода, причем очень трудно доказать, что полученный программный комплекс эквивалентен исходному. На полное тестирование результирующей системы уходят годы, так как вместо исходного комплекса получается абсолютно другой, новый программный продукт, со своими ошибками и недосмотрами. Поэтому никто не сможет гарантировать, что учтены все тонкости исходной программы. Во-вторых, в таких случаях вместо программы на новом языке обычно получается та же программа на COBOL, но записанная в терминах C++. В-третьих, очень тяжело восстановить исходные знания, заложенные в систему. Собственно, при таком подходе это и не входит в задачу разработчиков. Главное — перевести на новый язык, а там уж разберемся! Но полученную систему очень тяжело поддерживать и развивать.

Фирм, использующих такой подход, очень много. Приведем несколько адресов: **http://www.aly-daar.com/**, **http://www.peritus.com/**, **http://www.keane.com/**

Другим вариантом, получившим распространение в последнее десятилетие, является метод восстановления структуры программы по исходному тексту с использованием различных CASE-средств. Такой процесс называют reverse engineering (см. **http://www.sei.cmu.edu/reengineering/** — сайт университета Карнеги-Мелона, США; где собрана неплохая подборка о теоретических основах реинжиниринга). Для использования такого метода требуется как минимум продукт, реализующий синтаксический анализ исходных программ.

Этот метод имеет свои очевидные преимущества, например, в результате вновь появляется утерянное с годами понимание смысла исходной программы. С другой стороны, это не слишком приближает появление конвертированной системы, так как и при этом подходе возможность прямой кодогенерации не предусмотрена, следовательно, сохраняются основные недостатки предыдущего пункта.

Тем не менее такой подход тоже достаточно распространен и дает неплохие результаты. В качестве примера назовем компании Viasoft (технология ESW — см. **http://www.viasoft.com/prdcts/esw/index.htm**) и CACI, успешно использовавшую собственную методологию RENovate (см. **http://www.caci.com**/) для выполнения различных проектов реинжиниринга.

Наконец, в последнее время появились средства, позволяющие автоматизировать весь процесс реинжиниринга, в том числе и генерацию целевого языка. Таких продуктов еще очень мало в связи с чрезвычайной сложностью предметной области и размытостью рамок задач, но их популярность неуклонно растет. Эти системы помогают сократить усилия по реинжинирингу системы в два и более раза! За счет чего возникает столь существенное сокращение объема работ?

Дело в том, что транслирующая система автоматически учитывает такие тонкости исходного приложения, которые человек при ручном реинжиниринге может попросту не заметить, — например, межмодульные взаимодействия. Иногда заказчикам это кажется необъяснимым фокусом, так как программы качественно переводятся без полного понимания человеком исходного программного комплекса! Но в большинстве случаев понимания и не требуется, к тому же оно стоит дороже. Если программа успешно работала в течение многих лет, то все, что нужно, — это перенести ее в новое окружение.

Внешне такие продукты выглядят как обычные компиляторы, с той лишь разницей, что вместо машинного кода они генерируют программу на другом языке: например, программы на COBOL преобразуются в программу на C++ или Java. Ясно, что задача такого необычного транслятора усложняется из-за принципиальных различий в исходном и целевом языках. Например, в Java нет оператора goto, а в COBOL такие

Терехов Андрей Николаевич, профессор, генеральный директор «ЛАНИТ-ТЕРКОМ».
Терехов Андрей Андреевич, научный сотрудник «ЛАНИТ-ТЕРКОМ»

операторы использовались сплошь и рядом. Поэтому для создания эквивалентной программы на целевом языке приходится производить весьма нетривиальные структурные преобразования программы.

В результате по исходному тексту на устаревшем языке строится программа на современном целевом языке. Чаще всего это неполная программа — в ней не хватает некоторых особо сложных фрагментов исходного кода; вместо них вставляются комментарии, содержащие непереведенные операторы. Таким образом, последнюю правку должен внести человек.

Поэтому такие продукты надо называть не трансляторами, а конвертерами. Разница между ними — в степени автоматизированности: транслятор переводит в машинный код весь исходный текст, а конвертер — от 80 до 90% (конечно же, чем больше, тем лучше).

Выигрыш состоит в том, что огромную часть рутинной работы берет на себя конвертер, а на долю человека остается небольшое количество преобразований. Мы имеем дело с ***хорошо автоматизированным*** процессом. Одним из очевидных преимуществ является существенное повышение качества кода на целевом языке.

Можно привести несколько примеров таких продуктов.

Одним из первых объявленных в этой области является ACM (Automated Cobol Migration) одноименной фирмы. Как гордо заявлено на сайте этой английской компании (см. **http://www.acm.co.uk**), для реинжиниринга используются последние достижения в области искусственного интеллекта — имеется в виду, что программы написаны на языке LISP...

К сожалению, эксперимент оказался неудачным — искусственный интеллект так и не смог приблизиться к человеческому. Продукт неудобен в использовании, так как имеет множество очевидных дефектов, например:

- отсутствие нормальной диагностики ошибок;
- небольшой процент переводимых конструкций исходных языков;
- нестабильность в работе.

Все это отпугивает потенциальных покупателей, поэтому на данный момент будущее системы ACM представляется весьма сомнительным.

Существует бесплатно распространяемая демоверсия.

Сравнительно недавно (в конце января 1998 г.) американская компания Metamorphic Corp. объявила о выпуске своего конвертера, способного переводить программы на COBOL в C++, Java и Visual Basic (см. **http://www.metamorphic.com/html/cobol.html**). Этот продукт хорошо оформлен и приятно выглядит. К сожалению, рассказать о нем в подробностях трудно — на Web-сервере представлено мало материалов.

Тем не менее можно назвать основные достоинства и недостатки этой системы. Основным достоинством является легкость в изучении продукта и его использовании. По утверждению авторов, «вы можете посадить за эту работу вашу секретаршу».

Но плюсы такого оптимистичного подхода превращаются в минусы, когда речь идет о реальном применении. Во-первых, этот конвертер переводит сравнительно небольшую часть исходных конструкций, пропуская все сколько-нибудь сложные. Во-вторых, то, что он умеет переводить, он переводит «в лоб», то есть без существенных преобразований исходной программы. Как следствие, вы получаете по сути ту же программу на COBOL, но записанную в терминах C++ или Java. Такие результаты очень сложно сопровождать и развивать. В-третьих, вообще не поддерживаются встроенные SQL- и CICS-операторы.

Более того, реальные системы состоят не только из программ на COBOL. Общепринятой практикой являлось использование программ на JCL (Job Control Language) для координации работ различных программ в рамках одного проекта, BMS — для описания интерфейса и общения с пользователем и т.д. Все это не учитывается в конвертере фирмы Metamorphic.

Таким образом, конвертер Cobol To C++, Java & VB хорош для перевода небольших программ на COBOL, но малопригоден для перевода реальных больших систем.

Одной из наиболее удачных попыток решения проблемы реинжиниринга приложений на COBOL является среда разработки приложений PERCobol фирмы Synkronix, выпущенная в марте 1998 года. Эта система (см. **http://www.svnkroni.com/**) занимает некоторое промежуточное место между компиляторами COBOL и средствами «чистого» реинжиниринга и потому могла бы быть рассмотрена и выше.

В основе системы PERCobol лежит уже описанная выше идея компиляции байт-кода языка Java (возможно, с последующей декомпиляцией текста на Java). В отличие от прочих компиляторов PERCobol использует только такую схему. Именно поэтому естественнее рассматривать PERCobol как средство сопровождения/развития/реинжиниринга.

Очевидны преимущества такого подхода:

- практически полная независимость от платформы;
- сравнительно небольшие денежные вложения;
- в случае наличия в организации большого количества программистов на COBOL отпадает необходимость переучивать их на новые технологии.

Интересно, что, несмотря на потенциальную возможность получения по исходному тексту на COBOL эквивалентного текста на Java, представители фирмы не рекомендуют отказываться от использования COBOL в качестве базового языка. Приведенные аргументы весьма разнообразны — необходимость переобучения персонала, сохранение привычной структуры приложений и т.д. Но реальные причины, скорее всего, в другом. Процитируем: «Несмотря на то что исходные тексты могут быть хорошо откомментированными и читабельными, результирующий поток управления в Java может

быть достаточно сложен, и, как следствие, труден для отслеживания средним программистом».

Откуда возникает такая сложность?

По-видимому, чрезмерная сложность полученной программы возникает из-за необходимости декомпилировать байт-код. Так как компиляция, вообще говоря, является процессом с потерей информации, то обратный процесс не может гарантировать полного структурного соответствия исходной программе на COBOL.

Именно поэтому логичнее производить реструктуризацию программы **до компиляции** в код, когда еще не утеряна информация о структуре программы (циклах, вызовах, условиях и пр.).

Кроме того, в связи с интерпретативной природой языка Java происходят потери в производительности (интерпретация байт-кода + потери на эмуляцию типов данных COBOL и некоторых специальных выражений, например, STRING/UNSTRING).

Наконец, возможно, самым существенным недостатком данной схемы является ее ограниченность рамками языка Java. На данный момент такие языки, как C++, Visual Basic и Delphi, в совокупности представляют значительно большую часть рынка, и поэтому замыкаться на Java совершенно неоправданно.

Несмотря на перечисленные выше недостатки, основные идеи системы PERCobol очень интересны.

Существует возможность получить демо-версию данного продукта.

Наконец, в начале февраля этого года фирма Relativity Technologies (**http://www.relativity.com**) объявила о начале продаж своего продукта RescueWare Workbench. Мы считаем, что на данный момент это лучший продукт в категории «автоматизированный реинжиниринг», поэтому опишем его несколько подробнее.

RescueWare Workbench представляет собой интегрированную среду реинжиниринга, учитывающую все аспекты исходной системы. Прежде всего для каждой программы производится ее глубокий анализ, включая ее взаимодействия с другими программами. RescueWare снабжена мощным набором средств визуальной декомпозиции исходной программы и потому может восприниматься и как средство reverse engineering'а. Но основной акцент делается не на этом.

В RescueWare процесс реижиниринга воспринимается как решение трех основных задач. Во-первых, это перенос данных из старых баз данных и «плоских» файлов в современные реляционные СУБД. Во-вторых, производится генерация современных GUI-интерфейсов на основе старых алфавитно-цифровых экранных форм. Наконец, в-третьих, производится конвертация самих программ с учетом описанных выше изменений. Общая схема работы продукта проиллюстрирована на рис. 1.

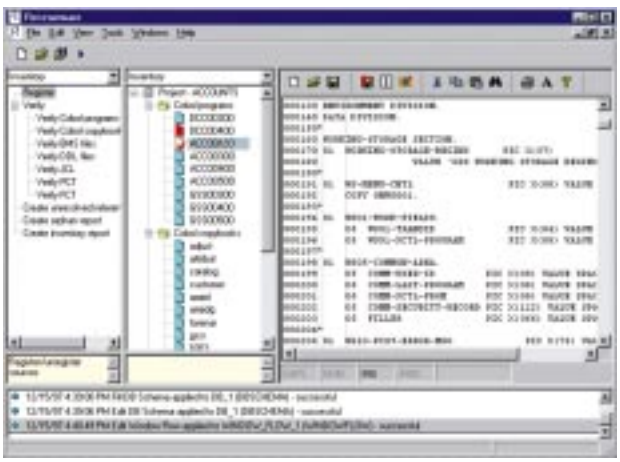

Рис. 2

На сайте фирмы также приводится типовой вид экрана во время работы с RescueWare (рис. 2).

В RescueWare есть также специальная интегрированная часть, помогающая решить проблему 2000 года.

Заметим, что сейчас практически все предприятия прилагают различные усилия для решения этой проблемы. Многие покупают программные продукты, помогающие только в решении «проблемы 2000 года», по сути выкидывая при этом деньги на ветер, так как такие решения не дают ничего, кроме исправления старой ошибки. Но более дальновидные руководители хотят убить сразу двух зайцев, вкладывая деньги в полноценный реинжиниринг существую-

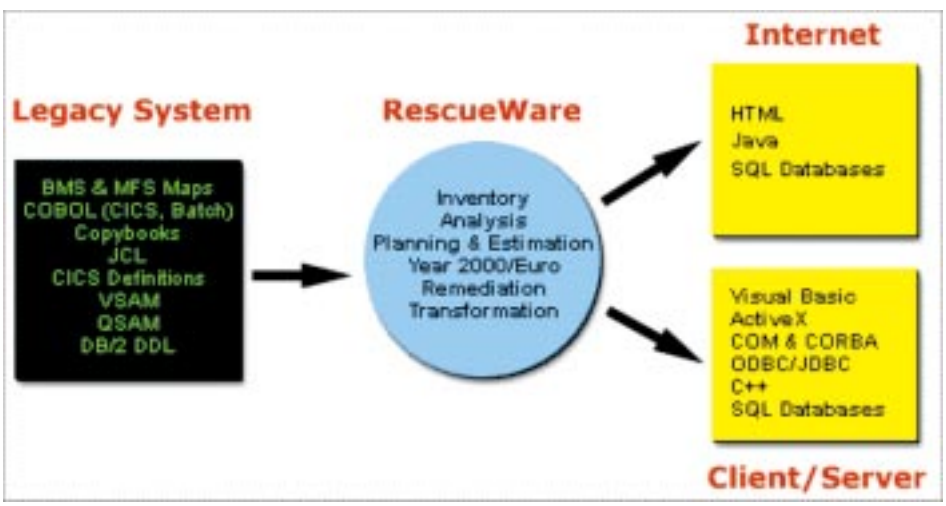

Рис. 1

щей системы, решая при этом и «проблему 2000 года», и проблему переноса системы в новые технологии.

Рынок программ, помогающих в решении «проблемы 2000 года», очень велик. Обзор продуктов, решающих «проблему 2000 года», вылился бы в значительно большую статью (см. http://www.year2000.com/)! Большинство таких программных продуктов представляют собой простые поисковые механизмы, ищущие в исходных текстах слова специального вида (например, Date, Year и т.п.) по специальным шаблонам. Но при использовании такого примитивного подхода невозможно определить все места, требующие исправления. Приведем простой пример: пусть в одной программе значение переменной X пишется в файл, а другая программа берет это значение из файла и записывает его в переменную Year. Очевидно, что обычный поиск по шаблону в таком случае не найдет всех «подозрительных» мест, так как не будет учтена переменная X. В RescueWare используется математический алгоритм, основанный на транзитивном замыкании множества подозрительных мест. Наконец, эквивалентное преобразование текста по указанным принципам (windowing, expansion и т.д.) производится автоматически (так называемый remediation).

В RescueWare производятся также различные оптимизационные просмотры, упрощающие программу. Это вызвано тем, что исходные системы развивались в течение долгого времени и поэтому в них часто возникали различные наслоения. Поэтому полученную программу обычно можно существенно улучшить — убрать недостижимый код, неиспользуемые переменные и т.п.

Подытожим сказанное. RescueWare Workbench является функционально полной интегрированной средой реинжиниринга и решения проблемы 2000 года, поддерживает более десяти различных диалектов COBOL, CICS и SQL-вставки, языки описания интерфейсов BMS и AS/400 screens. RescueWare включает мощные средства анализа и визуальной декомпозиции исходных программ. В результате работы RescueWare создает программные комплексы, использующие следующие языки и форматы:

- различные современные языки программирования (C++, Java и Visual Basic);
- структуры таблиц современных реляционных баз данных (Oracle, Sybase и прочие), поддерживаются также стандарты ODBC и JDBC;
- современный GUI-интерфейс (HTML, Java, Visual Basic, C++);
- поддерживаются стандарты ActiveX и CORBA.

Таким образом, на данный момент RescueWare Workbench является наиболее полным продуктом, предоставляющим автоматизированное решение проблем реинжиниринга и 2000 года. На сайте Relativity Technologies несколько раз упомянут тот факт, что RescueWare Workbench целиком и полностью создан группой программистов из Санкт-Петербургского и Новосибирского Государственных университетов.