Автоматизация процесса миграции данных с помощью *Python*

Rainer Fischbach, Ingrid Lachmann, Mario Winnemuth (ECS GmbH, Германия)

– How could God create the world within six days?
– There was no installed base!

Как Богу удалось создать мир в шесть дней? – спрашивается в старой *IT*-шутке. Ответ прост: не мешала старая инсталляция!

Этот юмористический диалог служит совершенно серьезным напоминанием сегодняшним *IT*-экспертам о безнадежности мечты начать внедрение с белого листа, с нуля, с самого начала. Реалии современного мира информационных технологий таковы, что там, где вводятся новые системы, обычно уже давно эксплуатируются старые, наследство которых и предстоит перенять. Сложность этой "само собой разумеющейся" задачи очень часто недооценивается, а объем требуемых затрат зачастую становится очевидным только на поздних стадиях проекта, когда бюджет уже на исходе и сроки, как говорится, крепко поджимают.

Некоторые проблемы миграции

Поскольку "с нуля" начать невозможно, рекомендуется параллельно с введением новой системы планировать миграцию (перенос) унаследованных данных. Обычно требуется полное их обновление, так как использовать данные в том виде, в котором они представлены в прежней системе, практически нельзя. Причина не только в том, что организация базы данных в старой системе, как правило, отличается (хотя уже одно это требует реорганизации в структуре классов, отношений и атрибутов). Унаследованные данные подвержены и ряду других недугов. Перечислим некоторые, наиболее типичные из них:

- идентичные обстоятельства могут быть описаны доброй дюжиной разных способов;
- одни и те же атрибуты в различных системах могут иметь разные допустимые значения;
- с точки зрения новой системы, данные старой системы обычно неполны;
- документы и процессы находятся в разной степени обработки, что дополняется наличием "мертвых душ" (данные в БД, не обрабатывавшиеся годами);
- одна и та же прикладная программа в разных подразделениях (филиалах) предприятия может работать с данными различного формата. Эти различия возникают вследствие недокументированных отклонений от когда-то определенной, но устаревшей схемы.

Осуществляя перенос данных, необходимо быть готовым практически ко всему. В особенности это касается *метаданных*. Здесь часто господствует оптимистический подход, исходящий из того, что в описанных документах нет отклонений от правил форматирования, индексирования и наименования файлов. Но ведь оригиналы – будь то документы в формате *Word*, будь то *CAD*-чертежи со стандартными текстовыми полями – терпеливы, как бумага... С другой стороны, метаданные, которые удается экстрагировать из старых документов (например, из переписки, заметок-напоминаний, чертежей) часто представляют собой кладезь информации для предприятия. Таким образом, системное извлечение и "очистка" старых данных, включая проверку на достоверность различных источников, имеет смысл по двум причинам: обеспечение практически безошибочной обработки данных в новой системе и ввод в действие до сих пор неиспользованного потенциала.

В этой ситуации необходима формализованная процедура обнаружения и протоколирования ошибок в данных, позволяющая позднее произвести очистку и изменение структуры данных.

Первый шаг к успеху в переносе старых данных состоит в том, что такие действия, как учет затрат на "ремонт" данных и оценка миграционного риска, производятся уже на стадии планирования проекта. В противном случае, когда миграция проводится в виде краткосрочной операции накануне сдачи новой системы, она способна взорвать все границы – как бюджетные, так и временны́е.

Для наглядности процесс миграции можно сравнить с наведением моста; модели данных старой и новой систем при этом находятся на разных берегах. О цели на другом берегу (то есть, о новой модели данных и их форматах) представление имеется достаточно четкое, а вот на исходном берегу почва довольно зыбкая, и, чтобы детально разобраться с имеющейся моделью данных, часто требуются инвестиции в её реконструкцию. Опросы пользователей и программистов ведут к успеху только после многократного повторения цикла "допущение – тест – новое допущение".

Рекомендуется промежуточный шаг

Не сто́ит пытаться сразу перевести старые данные непосредственно в новый формат, так как поначалу трудно оценить, какие затраты потребуются на их очистку и ремонт. К тому же, при этом могут быть потеряны возможности унификации процессов и инструментов.

Гораздо более разумным шагом будет, продолжая аналогию с мостом, соорудить промежуточную опору (в нашем случае – промежуточный формат). К такому промежуточному формату легко применять универсальные инструменты обработки данных. При помощи стандартизированных процессов и инструментов можно аккуратно провести шаги по очистке и реорганизации данных. Отсюда будет уже легко перейти к целевому формату.

Наиболее важными критериями качества миграции и "ремонта" являются, наряду с безошибочностью данных, их воспроизводимость и изменяемость. Первой предпосылкой для этого является документированный процесс с точно определенными шагами и промежуточными результатами, которые можно подвергнуть перепроверке.

Целенаправленная автоматизация

Следующее требование, исходящее из объемов унаследованных данных, состоит в максимально далеко идущей автоматизации процессов. Обрабатывать большие объемы данных вручную (например, используя команду "найти-заменить") означает подвергаться риску внесения непоправимых ошибок. Это неприемлемо, даже если не учитывать то, что скорость таких процедур и их функциональность в части распознавания и замены текстовых шаблонов явно недостаточны.

Для последовательной автоматизации требуются программируемые, параметризируемые и конфигурируемые инструменты. Достаточно эффективными показали себя скриптовые языки, содержащие все функции обработки текстов – сегодня это *Perl* и *Python*. Оба являются полноценными интерпретируемыми языками программирования с усложненными типами данных (например, *string*, список, словарь), оба имеют функции для распознавания и замены фрагментов текста по образцу.

В пользу *Python* говорят такие факторы, как большая доступность, мобильность, интегрируемость с *C/C++*, качество проектирования программного обеспечения, надежная обработка текстов и легкость в освоении. Имеется интерпретатор *Python* для множества платформ – *Windows*; различных производных от *UNIX* и *Linux*; систем, базирующихся на *BSD*, *OS/2*, а также для встраиваемых систем реального времени *QNX* и *Vx Works*.

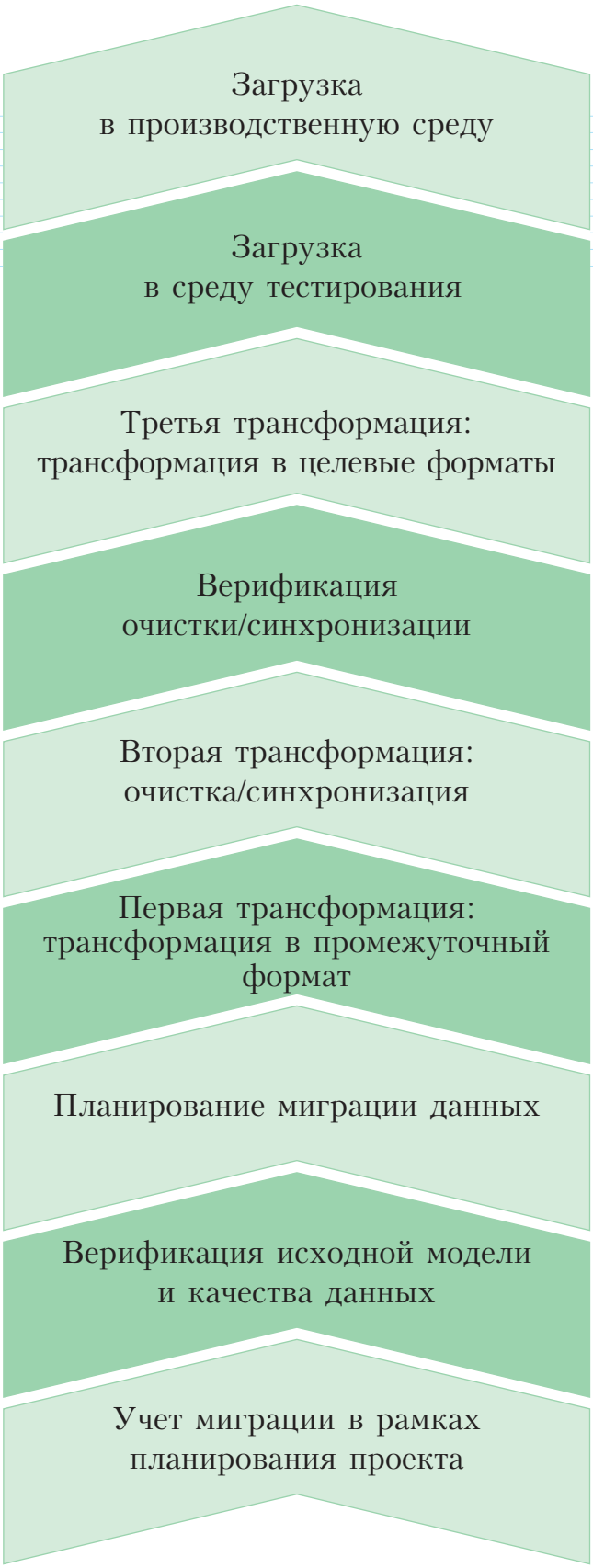

Рис. 1. Основные этапы переноса унаследованных данных в новую IT-систему

К этому можно добавить поддержку *Apple* и "наладонников" с *Palm OS*, *Epoc* и *Windows CE*. Однотипные интерфейсы к системным функциям позволяют использовать идентичные *Python*-программы для разных платформ. Обеспечивается интеграция с приложениями и библиотеками, написанными в *C/C++* или *Java*. Вызов функций (методов) возможен в обоих направлениях.

Четкая концепция

Язык *Python* имеет четкую концепцию, которая обеспечивает возможность качественного проектирования программного обеспечения. Комбинация простых семантических понятий, логичный синтакс, подразделение кода на семантические и синтаксические единицы, образующие ступенчатую структуру, а также мощные стандартные типы данных – всё это позволяет в короткое время создавать компактные, легко читаемые программы для относительно сложных задач.

Что касается надежной обработки текстов, то *Python* позволяет так организовать регулярные выражения для анализа текстов и их систематического изменения (в том числе и в *Unicode*), что процесс написания и ухода за программным кодом, работающим с этими выражениями, значительно упрощается. Помимо этого, исключаются типичные ошибки, возникающие при изменениях в регулярных выражениях.

Наконец, весомыми аргументами в пользу *Python* являются небольшое количество и простота правил, "прозрачность" кода. Это позволяет овладеть работой с этим языком программирования в значительно более короткий срок, чем того требуют конкурирующие языки.

Как следствие, *Python* стал не только языком для "одноразовых" скриптов, но и средством создания универсальных инструментов. В качестве примера можно назвать конфигурируемый фильтр для таблиц данных, который применятся в процессе избавления от устаревшей или разнородной системы обозначений в унаследованных данных. Необходимые замены в каждом поле при этом определяются с помощью таблиц или регулярных выражений.

Подводя итог, можно сказать, что перенос старых данных в новые *IT*-системы требует значительных усилий. Скриптовый язык *Python* оптимально приспособлен для обеспечения автоматической трансформации данных в промежуточный формат. Это одна из реальных возможностей повысить эффективность процесса миграции.