Н.С. Макаров

UML: поддержка проектирования и инструментальные среды

В статье рассмотрены инструменты поддержки проектирования и использования шаблонов при разработке прикладных программных продуктов. Особое внимание уделено возможностям моделирования и описания предметной области, а также применению CASE-инструментов для упрощения процесса трансляции моделей отдельных элементов и проекта в целом формализованные описания для генерации программного кода. Автор рассматривает основные аспекты моделирования как с индивидуальной, так и с коллективной точки зрения поддержки разработки проектов.

Развитие CASE-средств шло поэтапно. Эволюция привела CASE-средства к высшему уровню в процессе разработки — проектированию и моделированию. Эти два процесса представляют собой наиболее ответственную стадию, трудоемкость и сложность которой невозможно переоценить. Моделирование предметной области и построение рабочей модели представляют собой первоисточник для всего проекта в целом.

Выделение сущностей в какой-либо предметной области до сих пор является процессом, основанным более на эмпирическом, нежели на формализованном знании. Аналитики, выполняющие эту работу, отражают предметную область в некоторой форме понятной для проектировщиков.

Передача результатов анализа предметной области означает формальный старт проекта. Однако как показывает практика, достаточно большое количество проектов ведется по итерационной схеме разработки, вследствие чего на каждом этапе требования к продукту уточняются, а предметная область дополняется и расширяется ее описание. Таким образом, незначительные дополнения осуществляемые в процессе моделирования зачастую могут вызвать трудоемкие изменения в самом проекте.

Именно использование CASE-средств на этапе моделирования и описания предметной области, может снизить трудозатраты на каждой итерации проекта. Для реализации этого необходимо внедрить на уровне моделирования предметной области инструмент, который можно было бы использовать на всех уровнях исполнения проекта, начиная с моделирования и заканчивая кодированием и тестированием.

В настоящее время единственным полнофункциональным средством моделирования с четкой и строго формализованной структурой является UML (Unified Modeling Language).

Почему UML?

Применение UML является наиболее существенным нововведением в подходах к разработке, основанной на моделировании, которое включает в себя нормальные процессы и создание эффективных архитектур по сравнению с предыдущими методиками и нотациями описания предметной области. Спецификация UML дает разработчику возможность использовать стандартную нотацию для моделирования систем, компонентов, поведения объектов и реакции пользователей. Спецификация рабочей группы Object Management Group для UML гласит, что унифицированный язык моделирования является графическим языком для визуализации, уточнения, конструирования и документирования артефактов

систем с интенсивным использованием программного обеспечения. UML предлагает стандартизованный способ написания прототипов систем, включающих такие вещи, как непосредственно программный код (выражения и функции), схемы потоков данных в базах данных (БД) и программные компоненты, предназначенные для многократного использования [1].

Основным в спецификации является то, что UML — это «язык» для определения того, чего способна будет достичь вся система, а не метод или процедура в решении каких-то конкретных проблем. Этот язык может быть использован для поддержки управления жизненным циклом программного обеспечения (ПО) самыми различными способами, однако изначально он был разработан именно для создания прототипов систем. Сначала проводится анализ требований к системе или программе, затем производится моделирование в UML и представляется решение первоначальной проблемы на концептуальном уровне. Спецификация UML не определяет конкретных средств реализации, а позволяет создать эскиз проблемы глазами аналитиков, чтобы затем идеи можно было легко передать команде разработчиков.

UML определяет нотацию и семантику для следующих типов решений проблем:

- модель взаимодействия ПО с пользователями системы — описывает взаимодействия между пользователями и системой, а также определяет границы этих отношений;
- модель взаимодействия и кооперации — описывает, как объекты системы взаимодействуют между собой для решения поставленных задач;
- динамическая модель — использует диаграммы для описания состояний, принимаемых классами с течением времени, и потока задач, которые система должна выполнять;
- логическая или классовая модель — описывает классы и объекты системы;
- модель материальных составляющих — описывает программное и иногда аппаратное обеспечение системы;
- модель материальной эксплуатации — описывает материальную (физическую) архитектуру и набор компонентов аппаратного обеспечения системы.

UML дает проекту возможность развития от простых приложений, ориентированных на решение конкретных пользовательских задач, до крупных программ, удовлетворяющих требованиям к системам индустриального масштаба. Разработчики получают в свое распоряжение элементы модели и указания для соответствия международным стандартам OMG (Object Management Group). Именно структуру процесса дает UML организации, занимающейся разработкой ПО. Такая структура позволяет ясно и четко увидеть сложные системные проблемы.

Тот факт, что UML «не является чей-то собственностью и открыт для всех» [1] позволяет использовать стандартную нотацию в целом ряде инструментов и языков, в том числе ориентированных на Open Source. UML доказал свою полезность, когда использовался Apache Group [2], и Common Object Request Broker Architecture (CORBA) [3]. UML дает возможность создания не зависящих от фирм-разработчиков спецификаций, которые могут быть использованы программистами, работающими под разными операционными системами.

Большинство CASE-средств поддерживают исключительно UML-методологию, которая является результатом творчества трех ведущих разработчиков: Гради Буча, Джеймса Рамбо и Айвара Якобсона. В отличие от других объектно-ориентированных методологий, UML задумывался как открытый стандарт для моделирования, который включил бы в себя достоинства многих других родственных разработок, развившихся за предыдущее время. Многие инструменты поддерживают UML версии 1.2 и позволяют пользователям переключаться между визуальными представлениями Буча, OMT (Ob-

ject Modeling Technology) и UML, для того чтобы помочь разработчикам в их миграции от более старых методов к UML.

Выбор инструментов для проектирования систем — это нелегкая задача, потому что большинство инструментов находятся на разных стадиях развития и только единицы из них позволяют решать сложные комплексные задачи. Разработка решения с одновременным использованием C, C++ и Java только усложнит задачу, потому что многие из продвинутых инструментов ориентированы только на один язык программирования или тип задач. Финансовые ограничения, которых придерживаются большинство разработчиков, приводят к созданию инструментов с ограниченной функциональностью, потому что отсутствие финансирования значительно снижает темпы и интенсивность развития инструмента.

Почему же большинство CASE-средств направленных на моделирование предметной области используют UML?

Причина в том, что UML определяет «семантическую метамодель», а не «модель для интерфейса», места для хранения, или запуска программ, хотя эти понятия могут быть достаточно близки [1]. Реалии *Open Source* таковы, что в большинстве случаев разработчики ограничиваются только определением основных функций продукта, а программы в итоге остаются недоделанными или плохо поддерживаются. Некоторые средства разработаны как кросс-платформенные, но такой подход имеет и другую сторону — приходится выбирать только Java в качестве языка для разработки. Остальные созданы специально для определенной операционной системы.

Недавние высказывания Мартина Фаулера о том, что природа конструирования архитектур умирает, не являются безосновательными, учитывая развитие экстремального программирования (*XP*). Экстремальное программирование — это процесс, возвращающий разработку программ к эволюционной разработке архитектуры вместо тщательно спланированного процесса. Экстремальное программирование — это способность сохранять ясность и простоту кода с сохранением неких образцов архитектуры, которые используют не постоянно, а только тогда, когда они становятся необходимыми.

Возможности CASE-средств

Средства моделирования позволяют графически проектировать систему, распространять дизайн проекта для широкой аудитории, публиковать модели и отчеты, а также генерировать рабочий код приложений и баз данных (Data Definition Language — DDL сценарии через Structured Query Language — SQL).

Вместе с тем, большинство средств позволяет генерировать модели из готового кода, что дает возможность проводить рефакторинг приложений на более высоком уровне.

Последние усовершенствования в UML допускают использовать семантику действий при описании модели, что, возможно, позволит полностью генерировать и управлять приложениями, используя UML (OMG называет это Model Driven Architecture — модельно-управляемой архитектурой).

Большинство средств моделирования попадает в одну из следующих групп:

- точечные решения — UML-средства моделирования, поддерживающие UML-нотацию и семантику, генерацию кода по диаграмме и генерацию диаграммы по коду, обычно интегрируются со средами разработки: *Rational Rose* (*IBM*), *Describe tool* (*Embarcadero*), *Magic Draw* (*No Magic, Inc.*);
- средства UML моделирования, которые поддерживают другие нотации моделирования: *System Architect* (*Popkin Software*);
- средства UML моделирования с встроенным многопользовательским репозиторием: *System Architect*, *UML Tau* (*Telelogic*), *AllFusion Component Modeler* (*Computer Associates*);
- средства UML моделирования, предназначенные для проектирования систем реального времени: *Rhapsody* (*I-Logix*);

• средства UML моделирования, интегрированные с кодом, вследствие чего они сами являются всего лишь другим представлением кода. Информация о модели хранится в виде кода. Диаграмма модели становиться еще и редактором кода: *TogetherSoft* (*Borland*);

• графические средства, позволяющие только рисовать диаграммы, без наполнения их содержанием.

Основные технические черты современных средств моделирования

Большинство CASE-средств отличаются своими техническими характеристиками и возможностями. Однако ввиду рассматриваемой специфики следует выделить основные возможности, которые необходимы для использования CASE-средств начиная со стадии моделирования и заканчивая кодированием.

Навигация

Современное CASE-средство должно поддерживать работу со множеством проектов, и удобную навигацию между ними. Также удобная навигация должна быть при управлении пакетами, классами и компонентами. Зачастую проекты настолько велики, что необходимо использовать утилиты поиска нужных классов и модулей, вследствие чего проблемы с удобной навигацией и интерфейсом становятся еще более острыми.

Хранилище (*репозиторий*)

В центре любого современного CASE-средства находится база данных с версиями проекта (модели, требований), содержащая наработки по проекту. Репозиторий выступает техническим центром проекта в контексте CASE-средств в целом — каждая версия проекта, вплоть до мельчайших классов, хранится в нем. В качестве базы данных для репозитория используется встроенная собственная база данных или промышленная БД (например, *Oracle*).

Репозиторий должен обладать следующими характеристиками:

• наличие браузера для удобной и быстрой навигации между версиями и проектами;

• возможность определения метамодели репозитория (его структуры);

• возможность получения, передачи данных из разных инструментов (документированный прикладной интерфейс программирования или открытую объектную модель);

• возможность ссылаться на внешние объекты (файлы или папки) или на другие объекты в репозитории;

• резервное копирование, восстановление, управление версиями объектов;

• возможность одновременной многопользовательской работы.

Расширяемость

Расширяемость подразумевает возможность совмещения инструмента моделирования с другими подобными инструментами или возможность захвата данных из различных источников. Она также включает в себя корректную реализацию метамодели UML, использование стереотипов. Расширяемость дает возможность команде разработчиков отойти от стандарта, либо изменить нотацию UML в соответствии с изучаемой ими предметной областью.

Совместная работа с моделями

Использование CASE-средств в больших проектах осложняется тем, что проект может быть достаточно территориально распределен. Поэтому CASE-средство должно предоставлять возможность совместной работы с моделью, проектом, репозиторием и прочими элементами. CASE-средство должно быть масштабируемым, так как с течением времени проект может разрастись, причем необходимо обеспечивать целостность проекта независимо от числа пользователей, а также контроль доступа и аудит всех изменений в репозитории.

Управление версиями и изменениями

Данная опция в большей степени касается руководящего состава проекта, неже-

ли рядовых пользователей, однако также должна обеспечиваться CASE-средствами.

Отчеты, отчетность, публикации

Документирование в процессе разработки является одной из наиболее трудоемких и неприятных функций. Современные CASE-средства автоматически генерируют проектную документацию, создают формы отчетов и бланки документов. Использование CASE-средств позволяет осуществлять публикацию отчетов по проектам с использованием открытых форматов (XML) и стандартов (HTML), делать проект открытым для широкой общественности, что особенно актуально для широко распространенных сейчас *Open Source* проектов.

Интеграция с продуктами других производителей

Функциональность CASE-средств достаточно широка, однако они не могут полностью заменить собой все вспомогательное программное обеспечение, используемое в процессе разработки. Поэтому в пределах проекта вместе с CASE-средствами используются и другие программные продукты, которые облегчают жизнь разработчиков.

Одной из стадий разработки является тестирование, которое обслуживается специализированными программными средствами. Для интеграции сре́ды тестирования и CASE-средства должны обладать открытым документированным интерфейсом доступа к функциям друг друга (Application Programming Interface — API).

Генерация кода по модели/модели по коду

Способность генерировать код по модели (code generation) и модель по коду (reverse engineering) позволяют сократить материальные, а самое главное временные затраты на кодирование.

Некоторые CASE-средства позволяют генерировать код на языках C++, C#, Java, Visual Basic. Однако UML дает семантическое описание, что не позволяет корректно реализовать код автоматизированными средствами. Генерация кода с помощью CASE-средств не отменяет труда программистов, хотя и существенно облегчает им жизнь, особенно в части кодирования примитивных классов. Использование средств генерации кода выводит CASE-средства на «последнюю милю» в проектных работах.

Таким образом, построив модель предметной области на первом этапе проекта, постепенно, на каждом последующем этапе разработки, уточняя и трактуя ее, проект доходит до стадии кодирования.

MDA

Идея MDA (Model Driver Architecture) заключается в том, чтобы генерировать код из модели. Модель описывается в UML, после чего представляется в платформо-независимом виде (Platform Independent Model — PIM). Далее модель может быть переведена в платформо-зависимую форму (Platform Specific Model — PSM). PSM содержит в себе дизайн и конкретную реализацию. Звено цепочки «PIM–PSM» представляет собой использование платформо-независимой модели в любых проектных целях, после чего с помощью нехитрых манипуляций PSM превращается в работающий код на каком-либо языке программирования.

PIM представляет собой точную спецификацию функций, структуры и поведения системы, чтобы транслироваться в PSM без потери семантики. Язык ограничений объектов обычно используется для описания ограничений модели в формализованном виде.

Использование шаблонов проектирования и семантики действий в нотации UML позволяют описать конкретное поведение объекта без определения деталей специфичных для конкретной платформы.

Таким образом, для полной поддержки MDA необходима поддержка UML нотации семантики действий и языка ограничения объектов.

XMI

XMI (eXtendable Markup Language Metadata Interchange) — XML обмена метаданны-

ми — это спецификация OMG, призванная обеспечить обмен информацией между средствами моделирования, поддерживая UML метамодель.

Обзор CASE-средств

JAM

Средство разработки приложений *JAM* (JYACC's Application Manager) — продукт фирмы *JYACC* (США). В настоящее время поставляется версия JAM 7 и готовится к выходу JAM 8.

Основной чертой JAM является его соответствие методологии RAD (Rapid Application Development), поскольку он позволяет достаточно быстро реализовать цикл разработки приложения, заключающийся в формировании очередной версии прототипа приложения с учетом требований, выявленных на предыдущем шаге, и предъявить его пользователю.

JAM имеет модульную структуру и состоит из следующих компонент:

- ядро системы;
- *JAM/DBi* (Data Base interface) — специализированные модули интерфейса к системе управления базой данных (СУБД) (*JAM/DBi-Oracle*, *JAM/DBi-Informix*, *JAM/DBi-ODBC* и т. д.);
- *JAM/RW* (Report Writer) — модуль генератора отчетов;
- *JAM/CASEi* (Computer Aided Software Engineering interface) — специализированные модули интерфейса к CASE-средствам (*JAM/CASE-TeamWork*, *JAM/CASE-Innovator* и т. д.);
- *JAM/TPi* (Transaction Processing interface) — специализированные модули интерфейса к менеджерам транзакций (например, *JAM/TPi-Server TUXEDO* и т. д.);
- *Jterm* — специализированный эмулятор X-терминала.

Ядро системы (собственно, сам JAM) является законченным продуктом, и может самостоятельно использоваться для разработки приложений. Все остальные модули — дополнительные, и самостоятельно использоваться не могут.

Ядро системы включает в себя следующие основные компоненты:

- редактор экранов: среда разработки экранов, визуальный репозиторий объектов, собственная СУБД JAM — JDB (JAM Data Base), менеджер транзакций, отладчик, редактор стилей;
- редактор меню;
- набор вспомогательных утилит;
- средства изготовления промышленной версии приложения.

При использовании JAM разработка внешнего интерфейса приложения представляет собой визуальное проектирование и сводится к созданию экранных форм путем размещения на них интерфейсных конструкций и к определению экранных полей ввода/вывода информации. Проектирование интерфейса в JAM осуществляется с помощью редактора экранов. Приложения, разработанные в JAM, имеют многооконный интерфейс. Разработка отдельного экрана заключается в размещении на нем интерфейсных элементов, возможной (но не обязательной) их группировке и конкретизации различных их свойств, включающих визуальные характеристики (позиция, размер, цвет, шрифт и т. п.), поведенческие характеристики (многообразные фильтры, форматы, защита от ввода и т. п.) и ряд свойств, ориентированных на работу с БД.

Редактор меню позволяет разрабатывать и отлаживать системы меню. В нем реализована возможность построения пиктографических меню (так называемые toolbar). Назначение каждого конкретного меню тому или иному объекту приложения осуществляется в редакторе экранов.

В ядро JAM встроена однопользовательская реляционная СУБД JDB. Основным назначением JDB является прототипирование приложений в тех случаях, когда работа со штатной СУБД невозможна или нецелесообразна. В JDB реализован необходимый

минимум возможностей реляционных СУБД за исключением индексов, хранимых процедур, триггеров и представлений (view). С помощью JDB можно построить БД, идентичную целевой (с точностью до отсутствующих в JDB возможностей) и разработать значительную часть приложения.

Отладчик позволяет проводить комплексную отладку разрабатываемого приложения. Осуществляется трассировка всех событий, возникающих в процессе исполнения приложения.

Утилиты JAM включают три группы:

- конверторы файлов экранов JAM в текстовые. JAM сохраняет экраны в виде двоичных файлов собственного формата. В ряде случаев (например, для изготовления программной документации проекта) необходимо текстовое описание экранов;
- конфигурирование устройств ввода/вывода. JAM и приложения, построенные с его помощью, не работают непосредственно с устройствами ввода/вывода. Вместо этого JAM обращается к логическим устройствам ввода/вывода (клавиатура, терминал, отчет). Отображение логических устройств в физические осуществляется с помощью средств конфигурирования;
- обслуживание библиотек экранов (традиционные операции с библиотеками).

Одним из дополнительных модулей JAM является генератор отчетов. Компоновка отчета производится в редакторе экранов JAM. Описание работы отчета осуществляется с помощью специального языка. Генератор отчетов позволяет определить данные, выводимые в отчет, группировку выводимой информации, форматирование вывода и др.

Приложения, разработанные с использованием JAM, не требуют так называемых исполнительных (run-time) систем и могут быть изготовлены в виде исполняемых модулей. Для этого разработчик должен иметь компилятор С и редактор связей. Для изготовления промышленной версии в состав JAM входит файл сборки (makefile), исходные тексты (на языке С) ряда модулей приложения и необходимые библиотеки.

JAM содержит встроенный язык программирования JPL (JAM Procedural Language), с помощью которого в случае необходимости можно написать модули, реализующие специфические действия. Данный язык является интерпретируемым, что упрощает отладку. Существует возможность обмена информацией между средой визуально построенного приложения и такими модулями. Кроме того, в JAM реализована возможность подключения внешних модулей, написанных на каком-либо языке, совместимом по вызовам функций с языком С.

С точки зрения реализации логики приложения JAM является событийно-ориентированной системой. В JAM определен набор событий, включающий открытие и закрытие окон, нажатие клавиш клавиатуры, срабатывание системного таймера, получение и передачу управления каждым элементом экрана. Разработчик реализует логику приложения путем определения обработчика каждого события. Например, обработчик события «нажатие кнопки на экране» (мышью или с помощью клавиатуры) может открыть следующее экранное окно. Обработчиками событий в JAM могут быть как встроенные функции JAM, так и функции, написанные разработчиком на С или JPL. Набор встроенных функций включает в себя более 200 функций различного назначения. Они доступны для вызовов из функций, написанных как на JPL, так и на С.

Промышленная версия приложения, разработанного с помощью JAM, включает в себя следующие компоненты:

- исполняемый модуль интерпретатора приложения. В этот модуль могут быть встроены функции, написанные разработчиками на языках третьего поколения;
- экраны, составляющие само приложение, могут поставляться в виде отдельных файлов, в составе библиотек экранов или же быть встроены в тело интерпретатора;

• внешние JPL-модули могут поставляться в форме текстовых файлов или в прекомпилированном виде. Причем прекомпилированные внешние JPL-модули могут быть как отдельными файлами, так и входить в состав библиотек экранов;

• файлы конфигурации приложения — файлы конфигурации клавиатуры и терминала, системных сообщений, общей конфигурации.

Взаимодействие с другими средствами

Непосредственное взаимодействие с СУБД реализуют модули *JAM/DBi*. Способы реализации взаимодействия в JAM разделяются на два класса: ручные и автоматические. При ручном способе разработчик приложения самостоятельно пишет запросы на SQL, в которых как источниками, так и адресатами приема результатов выполнения запроса могут быть интерфейсные элементы визуально спроектированного внешнего уровня, или внутренние, невидимые для конечного пользователя переменные. Автоматический режим, реализуемый менеджером транзакций JAM, осуществим для типовых и наиболее распространенных видов операций с БД, так называемых QBE (Query By Example — запросы по образцу), с учетом достаточно сложных взаимосвязей между таблицами БД и автоматическим управлением атрибутами экранных полей ввода/вывода в зависимости от вида транзакции (чтение, запись и т. д.), в которой участвует сгенерированный запрос.

JAM позволяет строить приложения для работы более чем с 20 СУБД: *Oracle*, *Informix*, *Sybase*, *Ingres*, *InterBase*, *NetWare SQL Server*, *Rdb*, *DB2*, *ODBC*-совместимые СУБД и др.

Отличительной чертой JAM является высокий уровень переносимости приложений между различными платформами (*MS DOS/MS Windows*, *SunOS*, *Solaris* (*i80x86, SPARC*), *HP-UX*, *AIX*, *VMS/Open VMS* и др.). Может потребоваться лишь «перерисовать» статические текстовые поля на экранах с русским текстом при переносе между средами *DOS–Windows–UNIX*. Кроме того, переносимость облегчается тем, что в JAM приложения разрабатываются для виртуальных устройств ввода/вывода, а не для физических. Таким образом, при переносе приложения с платформы на платформу, как правило, требуется лишь определить соответствие между физическими устройствами ввода/вывода и их логическими представлениями для приложения.

Использование SQL в качестве средства взаимодействия с СУБД также создает предпосылки для обеспечения переносимости между СУБД. При условии переноса структуры самой БД в ряде случаев приложения могут не требовать модификации, за исключением инициализации сеанса работы. Такая ситуация может сложиться в том случае, если в приложении не использовались специфические для той или иной СУБД расширения SQL.

При росте нагрузки на систему и сложности решаемых задач (распределенность и гетерогенность используемых ресурсов, количество одновременно подключенных пользователей, сложность логики приложения) применяется трехзвенная модель архитектуры «клиент-сервер» с использованием менеджеров транзакций. Компоненты *JAM/TPi-Client* и *JAM/TPi-Server* позволяют достаточно просто перейти на трехзвенную модель. При этом ключевую роль играет модуль *JAM/TPi-Server*, так как основная трудность внедрения трехзвенной модели заключается в реализации логики приложения в сервисах менеджеров транзакций.

Интерфейс *JAM/CASEi* подобен интерфейсу к СУБД и позволяет осуществить обмен информацией между репозиторием объектов JAM и репозиторием CASE-средства аналогично тому, как структура БД импортируется в репозиторий JAM непосредственно из БД. Отличие заключается в том, что в случае интерфейса к CASE этот обмен является двунаправленным. Кроме модулей *JAM/CASEi*, существует также модуль *JAM/CASEi Developer's Kit*. С помощью этого мо-

дуля можно самостоятельно разработать интерфейс (т. е. специализированный модуль *JAM/CASEi*) для конкретного CASE-средства, если готового модуля *JAM/CASEi* для него не существует.

Мост (интерфейс) Silverrun-RDM JAM реализует взаимодействие между CASE-средством *Silverrun* и *JAM* (перенос схемы базы данных и экранных форм приложения между CASE-средством *Silverrun-RDM* и *JAM версии 7.0*). Данный программный продукт имеет 2 основных группы функций:

1. Прямой режим (Silverrun-RDM–JAM) предназначен для создания объектов CASE-словаря и элементов репозитория JAM на основе представления схем в Silverrun-RDM. В этом режиме мост позволяет, исходя из представления моделей данных интерфейса в Silverrun-RDM, производить генерацию экранов и элементов репозитория JAM. Мост преобразует таблицы и отношения реляционных схем RDM в последовательность объектов JAM соответствующих типов. Методика построения моделей данных интерфейса в Silverrun-RDM предполагает применение механизма подсхем для прототипирования экранов приложения. По описанию каждой из подсхем RDM мост генерирует экранную форму JAM; обратный режим (JAM–Silverrun-RDM) предназначен для переноса модификаций объектов CASE-словаря в реляционную модель Silverrun-RDM.

2. Режим реинжиниринга позволяет переносить модификации всех свойств экранов JAM, импортированных ранее из RDM, в схему Silverrun. На этом этапе для контроля целостности базы данных не допускаются изменения схемы через добавление или удаление таблиц и полей таблиц.

Ядро JAM имеет встроенный интерфейс к средствам конфигурационного управления (Polytron Version Control System — PVCS на платформе *Windows* и Source Code Control System — SCCS на платформе *UNIX*). Под управлением этих систем передаются библиотеки экранов и/или репозитории. При отсутствии таких систем JAM самостоятельно реализует часть функций поддержки групповой разработки.

Использование PVCS является более предпочтительным по сравнению с SCCS, так как позволяет организовать единый архив модулей проекта для всех платформ. Так как JAM на платформе *UNIX* не имеет прямого интерфейса к архивам PVCS, то выборка модулей из архива и возврат их в архив производятся с использованием PVCS Version Manager. На платформе *MS Windows* JAM имеет встроенный интерфейс к PVCS и действия по выборке/возврату производятся непосредственно из среды JAM.

JAM, как среда разработки, и приложения, построенные с его использованием, не являются ресурсоемкими системами.

Uniface

Uniface 6.1 — продукт фирмы *Compuware* (США), представляет собой среду разработки крупномасштабных приложений в архитектуре «клиент-сервер» и имеет следующие компоненты:

- *Application Objects Repository* (репозиторий объектов приложений) содержит метаданные, автоматически используемые всеми остальными компонентами на протяжении жизненного цикла информационной системы (ИС): прикладные модели, описания данных, бизнес-правил, экранных форм, глобальных объектов и шаблонов. Репозиторий может храниться в любой из баз данных, поддерживаемых *Uniface*;
- *Application Model Manager* поддерживает прикладные модели (E-R модели), каждая из которых представляет собой подмножество общей схемы БД с точки зрения данного приложения, и включает соответствующий графический редактор;
- *Rapid Application Builder* — средство быстрого создания экранных форм и отчетов на базе объектов прикладной модели. Оно включает графический редактор форм, средства прототипирования, отладки, тестирования и документирования. Реализо-

ван интерфейс с разнообразными типами оконных элементов управления (Open Widget Interface) для существующих графических интерфейсов — *MS Windows* (включая *VBX*), *Motif*, *OS/2*. Универсальный интерфейс представления (Universal Presentation Interface) позволяет использовать одну и ту же версию приложения в среде различных графических интерфейсов без изменения программного кода;

- *Developer Services* (службы разработчика) — используются для поддержки крупных проектов и реализуют контроль версий (Uniface Version Control System), права доступа (разграничение полномочий), глобальные модификации и т. д. Это обеспечивает разработчиков средствами параллельного проектирования, входного и выходного контроля, поиска, просмотра, поддержки и выдачи отчетов по данным системы контроля версий;
- *Deployment Manager* (управление распространением приложений) — средства, позволяющие подготовить созданное приложение для распространения, а также устанавливать и сопровождать его (при этом платформа пользователя может отличаться от платформы разработчика). В их состав входят сетевые драйверы и драйверы СУБД, сервер приложений (полисервер), средства распространения приложений и управления базами данных. *Uniface* поддерживает интерфейс практически со всеми известными программно-аппаратными платформами, СУБД, CASE-средствами, сетевыми протоколами и менеджерами транзакций;
- *Personal Series* (персональные средства) — используются для создания сложных запросов и отчетов в графической форме (Personal Query и Personal Access — PQ/PA), а также для переноса данных в такие системы, как *WinWord* и *Excel*;
- *Distributed Computing Manager* — средство интеграции с менеджерами транзакций *Tuxedo*, *Encina*, *CICS*, *OSF DCE*.

Объявленная в конце 1996 года версия *Uniface 7* полностью поддерживает распределенную модель вычислений и трехзвенную архитектуру «клиент-сервер» (с возможностью изменения схемы декомпозиции приложений на этапе исполнения). Приложения, создаваемые с помощью *Uniface 7*, могут применяться в гетерогенных операционных средах, использующих различные сетевые протоколы, одновременно на нескольких разнородных платформах (в том числе и в Internet).

В состав компонент *Uniface 7* входят:

- *Uniface Application Server* — сервер приложений для распределенных систем;
- *WebEnabler* — серверное ПО для эксплуатации приложений в Internet и Intranet;
- *Name Server* — серверное ПО, обеспечивающее использование распределенных прикладных ресурсов;
- *PolyServer* — средство доступа к данным и интеграции различных систем.

В список поддерживаемых СУБД входят *DB2*, *VSAM* и *IMS*; *PolyServer* обеспечивает также взаимодействие с операционной системой *MVS*.

Среда функционирования *Uniface* — все основные платформы *UNIX* и *MS Windows*.

Silverrun

CASE-средство *Silverrun* американской фирмы *Computer Systems Advisers, Inc.* (CSA) используется для анализа и проектирования информационных систем бизнес-класса и ориентировано в большей степени на спиральную модель жизненного цикла. Оно применимо для поддержки любой методологии, основанной на раздельном построении функциональной и информационной моделей (диаграмм потоков данных и диаграмм «сущность-связь»).

Настройка на конкретную методологию обеспечивается выбором требуемой графической нотации моделей и набора правил проверки проектных спецификаций. В системе имеются готовые настройки для наиболее распространенных методологий: DATARUN (основная методология, поддер-

живаемая *Silverrun*), Gane/Sarson, Yourdon/DeMarco, Merise, Ward/Mellor, Information Engineering. Для каждого понятия, введенного в проекте, имеется возможность добавления собственных описателей. Архитектура *Silverrun* позволяет наращивать среду разработки по мере необходимости.

Silverrun имеет модульную структуру и состоит из четырех модулей, каждый из которых является самостоятельным продуктом и может приобретаться и использоваться без связи с остальными модулями.

Модуль построения моделей бизнес-процессов в форме диаграмм потоков данных (Business Process Modeler — BPM) позволяет моделировать функционирование обследуемой организации или создаваемой информационной системы. В модуле BPM обеспечена возможность работы с моделями большой сложности: автоматическая перенумерация, работа с деревом процессов (включая визуальное перетаскивание ветвей), отсоединение и присоединение частей модели для коллективной разработки. Диаграммы могут изображаться в нескольких предопределенных нотациях, включая Yourdon/DeMarco и Gane/Sarson. Имеется также возможность создавать собственные нотации, в том числе добавлять в число изображаемых на схеме дескрипторов определенные пользователем поля.

Модуль концептуального моделирования данных ERX (Entity Relationship eXpert) обеспечивает построение моделей данных «сущность-связь», не привязанных к конкретной реализации. Этот модуль имеет встроенную экспертную систему, позволяющую создать корректную нормализованную модель данных посредством ответов на содержательные вопросы о взаимосвязи данных. Возможно автоматическое построение модели данных из описаний структур данных. Анализ функциональных зависимостей атрибутов дает возможность проверить соответствие модели требованиям третьей нормальной формы и обеспечить их выполнение. Проверенная модель передается в модуль RDM (Relational Data Modeler).

Модуль реляционного моделирования RDM позволяет создавать детализированные модели «сущность-связь», предназначенные для реализации в реляционной базе данных. В этом модуле документируются все конструкции, связанные с построением базы данных: индексы, триггеры, хранимые процедуры и т. д. Гибкая изменяемая нотация и расширяемость репозитория позволяют работать по любой методологии. Возможность создавать подсхемы соответствует подходу ANSI SPARC к представлению схемы базы данных. На языке подсхем моделируются как узлы распределенной обработки, так и пользовательские представления. Этот модуль обеспечивает проектирование и полное документирование реляционных баз данных.

Менеджер репозитория рабочей группы (Workgroup Repository Manager — WRM) применяется как словарь данных для хранения общей по всем моделям информации, а также обеспечивает интеграцию модулей *Silverrun* в единую среду проектирования.

Платой за высокую гибкость и разнообразие изобразительных средств построения моделей является такой недостаток *Silverrun*, как отсутствие жесткого взаимного контроля между компонентами различных моделей (например, возможности автоматического распространения изменений между DFD различных уровней декомпозиции). Следует, однако, отметить, что этот недостаток может иметь существенное значение только в случае использования каскадной модели жизненного цикла программного обеспечения.

Для автоматической генерации схем баз данных у *Silverrun* существуют мосты к наиболее распространенным СУБД: *Oracle*, *Informix*, *DB2*, *Ingres*, *Progress*, *SQL Server*, *SQLBase*, *Sybase*. Для передачи данных в средства разработки приложений имеются мосты к языкам 4GL: JAM, PowerBuilder, SQL Windows, Uniface, NewEra, Delphi. Все мосты позволяют загрузить в *Silverrun RDM* информацию из каталогов соответствующих

СУБД или языков 4GL. Это дает возможность документировать, перепроектировать или переносить на новые платформы уже находящиеся в эксплуатации базы данных и прикладные системы. При использовании моста *Silverrun* расширяет свой внутренний репозиторий специфичными для целевой системы атрибутами. После определения значений этих атрибутов генератор приложений переносит их во внутренний каталог среды разработки или использует при генерации кода на языке SQL. Таким образом, можно полностью определить ядро базы данных с использованием всех возможностей конкретной СУБД: триггеров, хранимых процедур, ограничений ссылочной целостности. При создании приложения на языке 4GL данные, перенесенные из репозитория *Silverrun*, используются либо для автоматической генерации интерфейсных объектов, либо для быстрого их создания вручную.

Для обмена данными с другими средствами автоматизации проектирования, создания специализированных процедур анализа и проверки проектных спецификаций, составления специализированных отчетов в соответствии с различными стандартами в системе *Silverrun* имеется три способа выдачи проектной информации во внешние файлы:

- система отчетов. Определив содержимое отчета по репозиторию, выдает отчет в текстовый файл. Этот файл можно затем загрузить в текстовый редактор или включить в другой отчет;
- система экспорта/импорта. Для более полного контроля над структурой файлов в системе экспорта/импорта имеется возможность определять не только содержимое экспортного файла, но и разделители записей, полей в записях, маркеры начала и конца текстовых полей. Файлы с указанной структурой можно не только формировать, но и загружать в репозиторий. Это дает возможность обмениваться данными с различными системами: другими CASE-средствами, СУБД, текстовыми редакторами и электронными таблицами;
- хранение репозитория во внешних файлах через ODBC-драйверы. Для доступа к данным репозитория из наиболее распространенных систем управления базами данных обеспечена возможность хранить всю проектную информацию непосредственно в формате этих СУБД.

Групповая работа поддерживается в системе *Silverrun* двумя способами:

- в стандартной однопользовательской версии имеется механизм контролируемого разделения и слияния моделей. Разделив модель на части, можно раздать их нескольким разработчикам. После детальной доработки модели объединяются в единые спецификации;
- сетевая версия *Silverrun* позволяет осуществлять одновременную групповую работу с моделями, хранящимися в сетевом репозитории на базе СУБД *Oracle*, *Sybase* или *Informix*. При этом несколько разработчиков могут работать с одной и той же моделью, так как блокировка объектов происходит на уровне отдельных элементов модели.

Имеются реализации *Silverrun* трех платформ — *MS Windows*, *Macintosh* и *OS/2 Presentation Manager* — с возможностью обмена проектными данными между ними.

Designer/2000 + Developer/2000

Designer/2000 2.0 фирмы *Oracle* является интегрированным CASE-средством, обеспечивающим в совокупности со средствами разработки приложений *Developer/2000* поддержку полного жизненного цикла программного обеспечения для систем, использующих СУБД *Oracle*.

Designer/2000 представляет собой семейство методологий и поддерживающих их программных продуктов. Базовая методология *Designer/2000* (*CASE*Method*) — структурная методология проектирования систем, полностью охватывающая все эта-

пы жизненного цикла ИС. В соответствии с этой методологией на этапе планирования определяются цели создания системы, приоритеты и ограничения, разрабатывается системная архитектура и план создания ИС. В процессе анализа строятся модель информационных потребностей (диаграмма «сущность-связь»), диаграмма функциональной иерархии (на основе функциональной декомпозиции ИС), матрица перекрестных ссылок и диаграмма потоков данных.

На этапе проектирования разрабатывается подробная архитектура ИС, проектируется схема реляционной БД и программные модули, устанавливаются перекрестные ссылки между компонентами ИС для анализа их взаимного влияния и контроля за изменениями.

На этапе реализации создается БД, строятся прикладные системы, проводится их тестирование, проверка качества и соответствия требованиям пользователей. Создается системная документация, материалы для обучения и руководства пользователей. На этапах эксплуатации и сопровождения анализируются производительность и целостность системы, выполняется поддержка и, при необходимости, модификация ИС.

Designer/2000 обеспечивает графический интерфейс при разработке различных моделей (диаграмм) предметной области. В процессе построения моделей информация о них заносится в репозиторий. В состав *Designer/2000* входят следующие компоненты:

- *Repository Administrator* — средства управления репозиторием (создание и удаление приложений, управление доступом к данным со стороны различных пользователей, экспорт и импорт данных);
- *Repository Object Navigator* — средства доступа к репозиторию, обеспечивающие многооконный объектно-ориентированный интерфейс доступа ко всем элементам репозитория;
- *Process Modeller* — средство анализа и моделирования деловой деятельности, основывающееся на концепциях реинжиниринга бизнес-процессов BPR и глобальной системы управления качеством TQM (Total Quality Management);
- *Systems Modeller* — набор средств построения функциональных и информационных моделей проектируемой ИС, включающий средства для построения диаграмм «сущность-связь», функциональных иерархий и потоков данных, а также средство анализа и модификации связей объектов репозитория различных типов;
- *Systems Designer* — набор средств проектирования ИС, включающий средство построения структуры реляционной базы данных (Data Diagrammer), а также средства построения диаграмм, отображающих взаимодействие с данными, иерархию, структуру и логику приложений, реализуемую хранимыми процедурами на языке PL/SQL (Module Data Diagrammer, Module Structure Diagrammer и Module Logic Navigator);
- *Server Generator* — генератор описаний объектов БД *Oracle* (таблиц, индексов, ключей, последовательностей и т. д.). Помимо продуктов *Oracle*, генерация и реинжиниринг БД может выполняться для СУБД *Informix*, *DB/2*, *Microsoft SQL Server*, *Sybase*, а также для стандарта ANSI SQL DDL и баз данных, доступ к которым реализуется посредством *ODBC*;
- *Forms Generator* (генератор приложений для *Oracle Forms*). Генерируемые приложения включают в себя различные экранные формы, средства контроля данных, проверки ограничений целостности и автоматические подсказки. Дальнейшая работа с приложением выполняется в среде *Developer/2000*;
- *Repository Reports* — генератор стандартных отчетов, интегрированный с *Oracle Reports* и позволяющий русифицировать отчеты, а также изменять структурное представление информации.

Репозиторий *Designer/2000* — хранилище всех проектных данных и может работать в многопользовательском режиме, обеспе-

чивая параллельное обновление информации несколькими разработчиками. В процессе проектирования автоматически поддерживаются перекрестные ссылки между объектами словаря и может генерироваться более 70 стандартных отчетов о моделируемой предметной области. Физическая среда хранения репозитория — база данных *Oracle*.

Генерация приложений, помимо продуктов *Oracle*, выполняется также для *Visual Basic*.

Designer/2000 можно интегрировать с другими средствами, используя открытый интерфейс приложений *API* (Application Programming Interface). Кроме того, с помощью *Oracle CASE Exchange* можно осуществлять экспорт/импорт объектов репозитория для обмена информацией с другими CASE-средствами.

Developer/2000 обеспечивает разработку переносимых приложений, внедренных в графической среде *Windows*, *Macintosh* или *Motif*. В среде *Windows* интеграция приложений *Developer/2000* с другими средствами реализуется через механизм OLE и управляющие элементы VBX. Взаимодействие приложений с другими СУБД (*DB/2*, *DB2/400*, *Rdb*) реализуется с помощью средств *Oracle Client Adapter* для *ODBC*, *Oracle Open Gateway* и *API*.

Среда функционирования *Designer/2000* и *Developer/2000 — Windows 3.x*, *Windows 95*, *Windows NT*, *Windows 2000* и т. д.

Vantage Team Builder (*Westmount I-CASE*)

Vantage Team Builder представляет собой интегрированный программный продукт, ориентированный на реализацию каскадной модели жизненного цикла и поддержку на протяжении всего этого цикла.

Он обеспечивает выполнение следующих функций:

- проектирование диаграмм потоков данных, «сущность-связь», структур данных, структурных схем программ и последовательностей экранных форм;
- проектирование диаграмм архитектуры системы — SAD — проектирование состава и связи вычислительных средств, распределения задач системы между вычислительными средствами, моделирование отношений типа «клиент-сервер», анализ использования менеджеров транзакций и особенностей функционирования систем в реальном времени;
- генерацию кода программ на языке 4GL целевой СУБД с полным обеспечением программной среды и генерацию SQL-кода для создания таблиц БД, индексов, ограничений целостности и хранимых процедур;
- программирование на языке C со встроенным SQL;
- управление версиями и конфигурацией проекта;
- многопользовательский доступ к репозиторию проекта;
- генерацию проектной документации по стандартным и индивидуальным шаблонам;
- экспорт и импорт данных проекта в формате CDIF (CASE Data Interchange Format).

Vantage Team Builder поставляется в различных конфигурациях в зависимости от используемых СУБД (*Oracle*, *Informix*, *Sybase* или *Ingres*) или средств разработки приложений (*Uniface*). Конфигурация *Vantage Team Builder for Uniface* отличается от остальных некоторой степенью ориентации на спиральную модель жизненного цикла ПО за счет возможностей быстрого прототипирования, предоставляемых *Uniface*. Для описания проекта ИС используется достаточно большой набор диаграмм, конкретные варианты которого для наиболее распространенных конфигураций приведены в табл. 1.

При построении всех типов диаграмм обеспечивается контроль соответствия моделей синтаксису используемых методов, а также контроль соответствия одноименных элементов и их типов для различных диаграмм.

При построении DFD обеспечивается контроль соответствия диаграмм различных уровней декомпозиции. Контроль за

Таблица 1

Типы диаграмм и их применение в различных СУБД

Тип диаграммы	Обозначение	*Oracle*	*Informix*	*Uniface*
Сущность-связь	ERD	+	+	+
Потоков данных	DFD	+	+	+
Структур данных	DSD	+	+	+
Архитектуры системы	SAD	+	+	+
Потоков управления	CSD	+	+	+
Типов данных	DTD	+	+	+
Структуры меню	MSD	+		
Последовательности блоков	BSD	+		
Последовательности форм	FSD		+	+
Содержимого форм	FCD		+	+
Переходов состояний	STD	+	+	+
Структурных схем	SCD	+	+	+

правильностью верхнего уровня DFD осуществляется с помощью матрицы списков событий (ELM). Для контроля за декомпозицией составных потоков данных используется несколько вариантов их описания: в виде диаграмм структур данных (DSD) или нотации БНФ (форма Бэкуса-Наура).

Для построения SAD используется расширенная нотация DFD, дающая возможность вводить понятия процессоров, задач и периферийных устройств, что обеспечивает наглядность проектных решений.

При построении модели данных в виде ERD выполняется ее нормализация и вводится определение физических имен элементов данных и таблиц, которые будут использоваться в процессе генерации физической схемы данных конкретной СУБД. Обеспечивается возможность определения альтернативных ключей сущностей и полей, составляющих дополнительные точки входа в таблицу (поля индексов), и мощности отношений между сущностями.

Наличие универсальной системы генерации кода, основанной на специфицированных средствах доступа к репозиторию проекта, позволяет поддерживать высокий уровень исполнения проектной дисциплины разработчиками: жесткий порядок формирования моделей, жесткую структуру и содержимое документации, автоматическую генерацию исходных кодов программ и т. д. — все это обеспечивает повышение качества и надежности разрабатываемых ИС.

Для подготовки проектной документации могут использоваться издательские системы *FrameMaker*, *Interleaf* или *Word Perfect*. Структура и состав проектной документации могут быть настроены в соответствии с заданными стандартами. Настройка выполняется без изменения проектных решений.

При разработке достаточно крупной ИС вся система в целом соответствует одному проекту как категории *Vantage Team Builder*. Проект может быть декомпозирован на ряд систем, каждая из которых представляется некоторой относительно автономной подсистемой ИС и разрабатывается независимо от других. В дальнейшем системы проекта могут быть интегрированы.

Процесс проектирования ИС с использованием *Vantage Team Builder* реализуется в виде четырех последовательных фаз (стадий): анализа, архитектуры, проектирования и реализации. При этом законченные результаты каждой стадии полностью или частично

переносятся (импортируются) в следующую фазу. Все диаграммы, кроме ERD, преобразуются в другой тип или изменяют вид в соответствии с особенностями текущей фазы. Так, DFD преобразуются в фазе архитектуры в SAD, DSD — в DTD. После завершения импорта логическая связь с предыдущей фазой разрывается, т. е. в диаграммы могут вноситься все необходимые изменения.

Конфигурация *Vantage Team Builder for Uniface* обеспечивает совместное использование двух систем в рамках единой технологической среды проектирования, при этом схемы БД (SQL-модели) переносятся в репозиторий *Uniface*, и, наоборот, прикладные модели, сформированные средствами *Uniface*, могут быть перенесены в репозиторий *Vantage Team Builder*. Возможные рассогласования между репозиториями двух систем устраняются с помощью специальной утилиты. Разработка экранных форм в среде *Uniface* выполняется на базе диаграмм последовательностей форм (FSD) после импорта SQL-модели. Технология разработки ИС на базе данной конфигурации показана на рис. 1.

Структура репозитория (хранящегося непосредственно в целевой СУБД) и интерфейсы *Vantage Team Builder* являются открытыми, что в принципе позволяет интеграцию с любыми другими средствами. *Vantage Team Builder* функционирует на всех основных *UNIX*-платформах (*Solaris*, *SCO UNIX*, *AIX*, *HP-UX*) и *VMS*. Ее используют в конфигурации «клиент-сервер», при этом база проектных данных может располагаться на сервере, а рабочие места разработчиков могут быть клиентами.

Локальные средства (*ERwin, BPwin, S-Designor, CASE.Аналитик*)

ERwin — средство концептуального моделирования БД, использующее методологию IDEF1X. *ERwin* реализует проектирова-

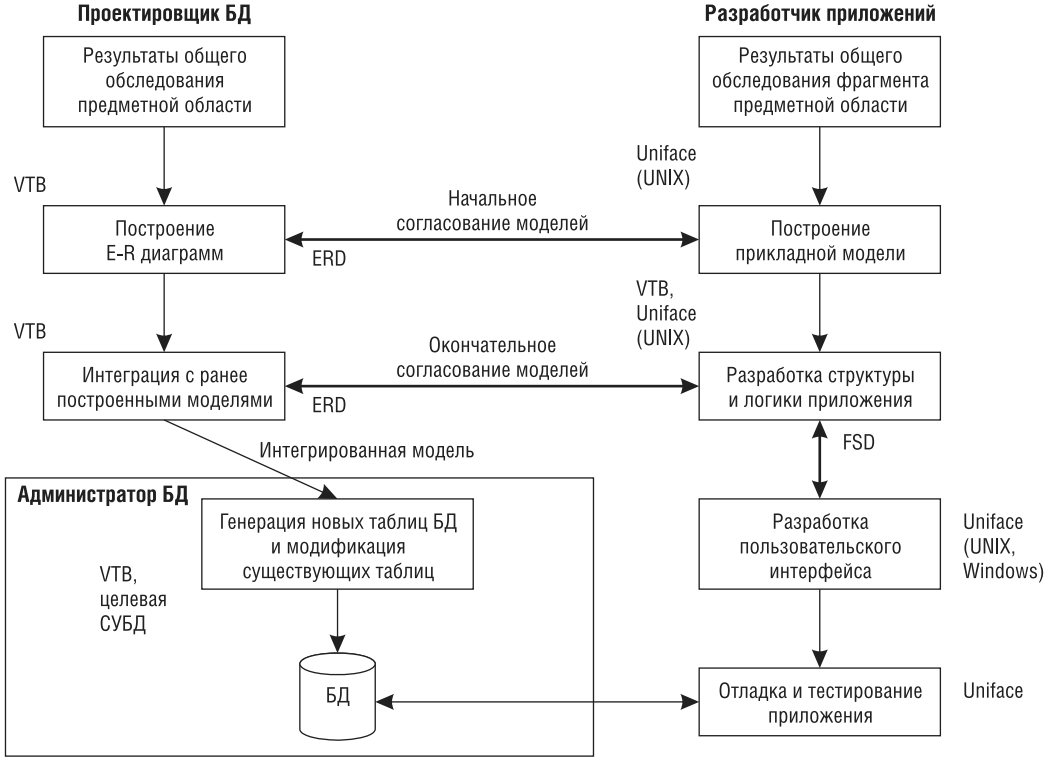

Рис. 1. Взаимодействие *Vantage Team Builder* и *Uniface*

ние схемы БД, генерацию ее описания на языке целевой СУБД (*Oracle*, *Informix*, *Ingres*, *Sybase*, *DB/2*, *Microsoft SQL Server*, *Progress* и др.) и реинжиниринг существующей БД. *ERwin* выпускается в нескольких различных конфигурациях, ориентированных на наиболее распространенные средства разработки приложений 4GL. Версия *ERwin/OPEN* полностью совместима со средствами разработки приложений *PowerBuilder* и *SQLWindows* и позволяет экспортировать описание спроектированной БД непосредственно в репозитории данных средств.

Для ряда средств разработки приложений (*PowerBuilder*, *SQLWindows*, *Delphi*, *Visual Basic*) выполняется генерация форм и прототипов приложений.

Сетевая версия *ERwin ModelMart* обеспечивает согласованное проектирование БД и приложений в рамках рабочей группы.

BPwin — средство функционального моделирования, реализующее методологию IDEF0.

S-Designor 4.2 представляет собой CASE-средство для проектирования реляционных баз данных. По своим функциональным возможностям и стоимости он близок к CASE-средству *ERwin*, отличаясь внешне используемой на диаграммах нотацией. *S-Designor* реализует стандартную методологию моделирования данных и генерирует описание БД для таких СУБД, как *Oracle*, *Informix*, *Ingres*, *Sybase*, *DB/2*, *Microsoft SQL Server*. Для существующих систем выполняется реинжиниринг БД.

S-Designor совместим с рядом средств разработки приложений (*PowerBuilder*, *Uniface*, *TeamWindows* и др.) и позволяет экспортировать описание БД в репозитории данных средств. Для *PowerBuilder* выполняется также прямая генерация шаблонов приложений.

CASE.Аналитик 1.1 является практически единственным в настоящее время конкурентоспособным отечественным CASE-средством функционального моделирования и реализует построение диаграмм потоков данных. Его основные функции:

- построение и редактирование DFD;
- анализ диаграмм и проектных спецификаций на полноту и непротиворечивость;
- получение разнообразных отчетов по проекту;
- генерация макетов документов в соответствии с требованиями ГОСТ 19.XXX и 34.XXX.

База данных проекта реализована в формате СУБД *Paradox* и является открытой для доступа.

С помощью отдельного программного продукта (*Catherine*) выполняется обмен данными с CASE-средством *ERwin*. При этом из проекта, выполненного в *CASE.Аналитике*, экспортируется описание структур данных и накопителей данных, которое по определенным правилам формирует описание сущностей и их атрибутов.

В заключение отметим, что интенсивное распространение CASE-инструментария, позволяющего интегрировать возможность проектирования с использованием UML и сред быстрой разработки приложений предоставляет разработчикам качественно новые возможности и существенно сокращает время разработки и внедрения программных проектов. Появление новых продуктов на рынке происходит достаточно быстро, поэтому данная статья является обзором широко распространенных продуктов и сред, и не претендует на обзор всех существующих инструментов.

Список литературы

1. *Буч Г., Рамбо Дж., Якобсон А.* Унифицированный язык моделирования. Руководство пользователя. Addison Wesley, 1998.

2. http://www.apache.org/.

3. http://www.corba.org/.

4. *Джилиам Дж.* Совершенствование модели Open Source с помощью UML и CASE-средств / Пер. с англ. П. В. Ступин // *Linux Gazette*. June 2001. № 67.

5. *Пендер Т.* Библия UML. John Wiley & Sons, 2003.

6. *Меллор С., Балсер М.* Прикладной UML — основы модельно-управляемой архитектуры. Addison Wesley, 2002.