УДК 338.364.2

Особенности программной реализации автоматизированной системы мониторинга и анализа «Графит»

МАЛАФЕЕВ А.В., инж., БЕЛОВ А.А., канд.техн.наук

При реализации программных продуктов зачастую возникает проблема совмещения типовых решений со спецификой и сложностью реального процесса. Представлен опыт разработки программного обеспечения автоматизированной системы мониторинга и анализа такого сложного процесса как процесс производства электрографитовых изделий (АСМА «Графит»)

АСМА «Графит» является клиент-серверным приложением. В роли сервера выступает сервер базы данных Firebird версии 1.5.1.4481. Для корректной работы необходимо использование суперсерверной реализации Firebird. Тип операционной системы для использования сервера может быть Windows или Linux, различия содержатся в модуле расширения ibs_udf. Для Windows – ibs_udf.dll, для Linux – ibs_udf.so. База данных без изменений может использоваться для обеих операционных систем. Для корректной работы клиентской части приложения рекомендуется использовать операционную систему MS Windows XP/2000.

Программный комплекс АСМА «Графит» является многопользовательской системой. Архитектура клиентского приложения представлена на рис. 1. Модульность программной реализации обеспечивается системными средствами Borland, заложенными в технологию пакетов bpl.

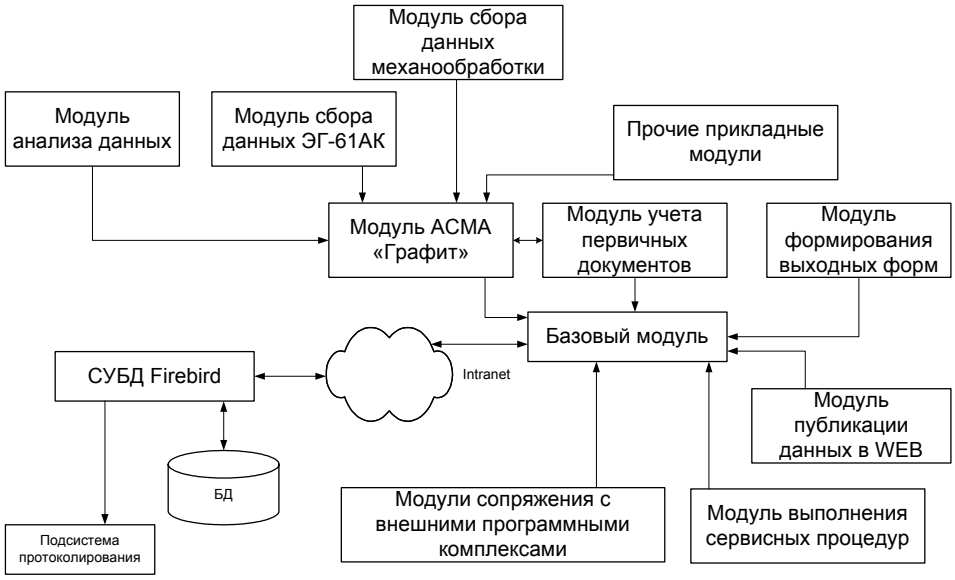

Рис. 1. Структура программной части АСМА «Графит».

Клиентское приложение является объектной оболочкой над реляционной базой данных. Все таблицы реляционной базы делятся на несколько групп:

1) системная часть;

2) технология – описание технологии производства;

3) учетные данные – параметрическое описание производственных циклов;

4) дополнительные таблицы производных подсистем.

Механизмы и способы расширения и развития программной системы. Основой программной реализации является технология Borland Package Library. Каждый модуль на представленной схеме (рис. 1), выделенный в отдельный bpl-пакет, основан на наборе пакетов ядра системы.

Ядро системы обеспечивает следующие функции:

- объектное представление информационной базы и базовый функционал информационных объектов;
- поддержку скриптовых подпрограмм уровня программных модулей, экранных форм, подсистемы формирования отчетов;
- автоматическую генерацию диалогов редактирования объектов, с поддержкой полей основных типов: строка, число, дата/время, логические значения, мемо-поля (в процессе развития системы существует возможность наследования форм, формирования расширенной функциональности форм редактирования отдельных объектов);
- контроль ссылочной целостности;
- дизайнер отчетов, библиотеку хранения и модуль генерации отчетов (выходных форм);
- библиотеку сервисных обработок;
- библиотеку запросов к базе данных;

● механизм протоколирования работы пользователей;
● управление пользователями, группами и доступом к информационным объектам;
● дизайнер пользовательского интерфейса.

Базовый модуль:
● поддержку объектного пространства;
● хранение структуры метаданных;
● формирование типовых форм редактирования данных;
● поддержку библиотек запросов, отчетов, скриптовых подпрограмм;
● разграничение прав пользователей;
● управление блокировкой объектов в многопользовательской системе;
● ведение журнала работы пользователей.

Основной модуль АСМА «Графит»:
● ведение основных справочников;
● формирование технологической карты и параметрического описания;
● типовые учетные формы операций, паспортов материалов, режимных карт, карт операций, кампаний, оборудования;
● формирование типовых отчетов;
● формирование маршрутных карт.

Модуль анализа данных:
● расчет показателей связности партий;
● формирование динамических OLAP кубов;
● корреляционный анализ;
● дисперсионный анализ.

Модуль сбора данных ЭГ61:
первичные формы сбора данных по технологии производства электрографитовых изделий марки ЭГ-61АК на подготовительном и технологическом этапах.

Модуль сбора данных механообработки:
поддержку первичных форм сбора данных по технологии производства электрографитовых изделий марки ЭГ-61АК на операциях механообработки.

Модуль учета первичных документов:
● настраиваемые первичные формы сбора данных;
● экспорт форм из пакета MS Excel;
● настраиваемый алгоритм фиксации форм в универсальную модель.

Модуль формирования выходных форм, отчетов:
● формирование типовых отчетов;
● редактор произвольных отчетов;
● экспорт отчетов в стандартные типы файлов (word, excel, pdf и прочее).

Модуль выполнения сервисных процедур:
● хранение и выполнение SQL подпрограмм обработки данных;
● хранение и выполнение скриптовых подпрограмм обработки данных.

Модуль сопряжения с внешними программными комплексами:
● скриптовые подпрограммы импорта данных;
● экспорт данных в пакет программ MS Office;
● экспорт данных в программу статистика;
● экспорт данных в типовые форматы: txt, cvs, dbf.

Модуль публикации данных в WEB:
● библиотеку исходных текстов;
● средства генерации содержания гипертекстовых документов;
● средства представления и печати гипертекстовых документов.

Приведенный список модулей является примером развития системы, где каждый последующий модуль конкретизирует и добавляет функциональность предыдущего. Данный тип расширения предполагает «жесткое» (изменение и расширение исходного кода) расширение функциональности с использованием средств разработки.

Многоуровневый принцип построения позволил выделить: постоянную часть, независимую от технологии производства, – технология и описание производственного процесса; переменную часть – формы ввода первичной информации на основе форм сбора данных, выходных форм, подпрограмм интеграции с внешними системами.

Для ввода первичных документов, которые обладают большой изменчивостью при переходе от одной технологии к другой, был разработан механизм табличных документов.

Табличный документ (макет) представляет собой электронную таблицу (рис. 2), подобно книге Excel, содержащую определенный набор данных, визуальное оформление, правила проверки, процедуру преобразования в структуру информационной базы. В системе может быть неограниченное количество макетов.

Документ представляет собой заполненный реальными данными макет. Документы объединяются в журналы документов. Для каждого макета задается основной журнал. Основной журнал определяет порядок нумерации документов. Также формируются дополнительные журналы, которые группируют документы по некоторым признакам.

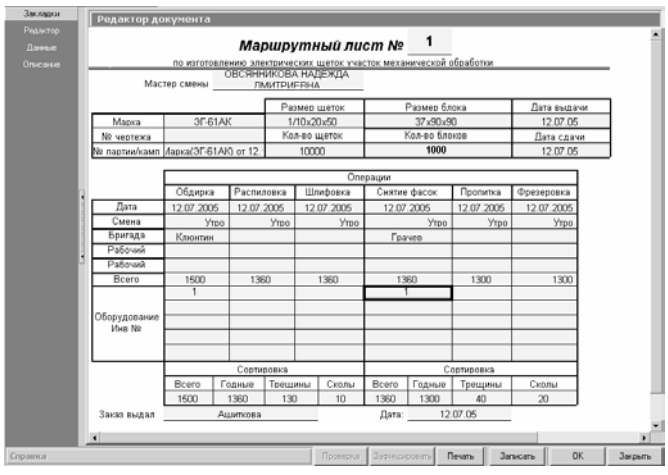

Рис. 2. Документ – Маршрутный лист. Операции механообработки.

Документ имеет одно из следующих состояний: созданный; записанный в информационную базу; помеченный на удаление; проверенный и зафиксированный. После создания, заполнения и сохранения документа в базу пользователь должен запустить процедуру проверки корректности документа, в случае неуспешной проверки пользователю выдается сообщение об ошибке заполнения и подсвечивается ячейка для корректировки данных. В случае успешной проверки пользователь фиксирует документ, тем самым выполняется скриптовая подпро-

грамма, которая на основе введенных данных вносит изменения в базу, например: создает кампанию, операцию; заполняет параметры и показатели определенной партии материала и пр.

Структура объектной системы. Основным понятием для понимания объектной структуры АСМА «Графит» является объектное пространство.

Объектное пространство представляет собой программный буфер (кэш), который накапливает происходящие в системе изменения, возникающие, например, при удалении, изменении или добавлении объектов, в том числе при реализации транзакционных операций. При этом специальные механизмы отслеживают накопление таких изменений и вычисляют «разницу» между состоянием элементов-объектов в памяти и состоянием уровня данных (базы данных). При вызове операции обновления уровня данных механизмы объектного пространства обеспечивают синхронизацию содержимого памяти и базы данных. В этом контексте объектное пространство реализует аналог известных механизмов каптируемого отложенного обновления, используемых при работе с СУБД.

Кроме того, объектное пространство содержит специальный механизм отображения данных, который служит своеобразным каналом между уровнем данных (реализуемым, например, посредством СУБД) и объектами, содержащимися в Object Space. Таким образом, в архитектуре приложения объектное пространство занимает центральную часть (рис. 3), реализуя следующие основные функции:

- хранение и представление информации о реализации элементов модели во время работы приложения; обеспечение целостной логической структуры приложения; обработку внутренних событий системы;
- отслеживание происходящих изменений в системе и формирование дельта-информации о накопленных различиях с уровнем данных;
- взаимодействие с графическим интерфейсом, предоставление информации о состоянии объектов;
- взаимодействие с уровнем данных, синхронизацию состояний памяти и базы данных.

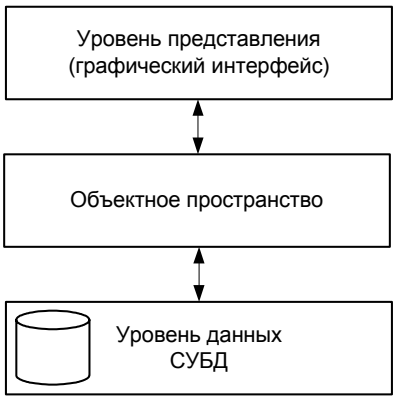

Рис. 3. Взаимодействие компонентов объектной системы

Для обеспечения таких функций реализованы специальные классы-носители информации, а также классы, реализующие управление элементами объектного пространства и осуществляющие обработку событий, происходящих в системе. Для иллюстрации решаемых на этом уровне системных задач рассмотрим вопрос контроля и управления жизненным циклом объектов во время работы приложения. При запуске приложения часть объектов загружается с уровня данных в объектное пространство (необходимость загрузки того или иного объекта может задавать разработчик на этапе создания/конфигурирования приложения или пользователь во время его выполнения). При этом часть объектов остается не загруженной в память, однако система считает, что они, тем не менее, существуют, поскольку не были удалены во время прошлых сеансов работы. Таким образом формируется совокупность объектов в памяти и объектов в БД, которая образует так называемое «концептуальное объектное пространство». Если происходит удаление объекта, он, тем не менее, остается в памяти и в БД, пока не будет вызван метод подтверждения транзакции. До момента этого вызова такой объект помечается системой как удаленный, но будет продолжать существовать как в памяти, так и в БД, однако он не будет существовать в концептуальном объектном пространстве и станет недоступным для использования.

Структура иерархии классов, обеспечивающая функционирование объектного пространства, довольно сложна и объемна и включает сотни классов. Мы рассмотрим только несколько ее наиболее важных фрагментов.

TRecordObject (TSimpleObject) – является центральным звеном в описании объектного пространства, суперклассом для всех его элементов. Он является универсальным классом, способным представлять все остальные объекты системы, сохраняющие свое состояние в базе. При генерации кода все пользовательские классы являются потомками TRecordObject. Все объекты в объектном пространстве также являются потомками этого класса либо дочерними по отношению к его потомкам.

TMainObjectBroker (TCustomObjectBroker) – представляет объектное пространство в целом. Основной задачей этого класса является поддержание соединения с базой данных, а также поддержка работы основной транзакции, через которую происходит чтение данных из базы.

IDatabaseNotificator – интерфейс обработки событий. Объекты классов, реализующих данный интерфейс, после регистрации подписки в потомке TCustomNotifyObject получают сообщения об изменении состояний объектов в системе, начале и окончании транзакций и другие события.

TRecordSet – класс, реализующий поддержку набора объектов, соответствующих одной таблице базы данных. Программа не знает, загружен объект в память или нет, до первого обращения к нему. При необходимости объект загружается. Данный класс определяет время жизни свободных объектов и выполняет периодическую выгрузку, освобождая тем самым оперативную память.

Остальные классы (TRecordSearch и TRecordCache), предназначенные для работы с наборами объектов, оперируют непосредственно только идентификаторами. Таким образом происходит экономия оперативной памяти и априорное разрешение конфликтов захвата объектов.

Класс TRecordSearch реализует оболочку для работы с наборами объектов. Он поддерживает процедуры поиска, сортировки и фильтрации объектов.

Класс TRecordCache является родителем для классов, реализующих подчиненные выборки объектов. Так, например, для объекта «Партия материала» реализуется доступ к набору технологических параметров, характеризующих данную партию, но являющихся в то же время объектами другого типа.

TLockObject – класс, реализующий механизм транзакций в объектной системе. С целью минимизации загрузки сервера БД в системе существует одна транзакция на чтение и неограниченное количество кратковременных транзакций на запись, которые активизируются лишь при необходимости переноса данных из объектного пространства в БД.

В отличие от технологии Bold, реализующей концепцию MDA, в АСМА «Графит» проектирование объектной модели начинается с построения реляционной базы данных, а также описания ее структуры в виде метаданных. Разработан генератор, который на основе информации о структуре БД и сформированных метаданных помодульно генерирует файлы классов – потомков базовых элементов.

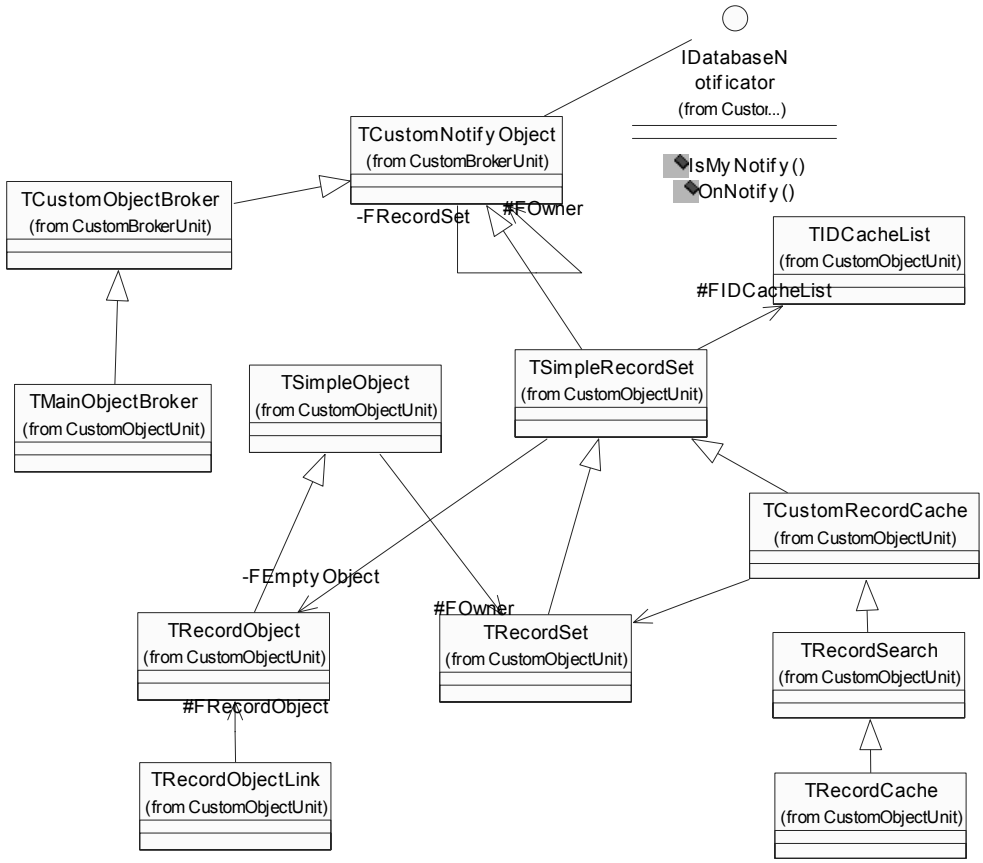

Рис. 4. Структура базовых объектов

Это позволяет программисту работать с объектами без необходимости отслеживать время жизни объекта, процесс его загрузки и сохранения в базе данных.

Типовой механизм работы с объектами позволил создать универсальные экранные формы, настраивающиеся в соответствии с описанием объекта в метаданных, но не ограничивающих разработчика. Базовый набор экранных форм позволяет быстро сформировать интерфейс работы с базой данных, а при необходимости скорректировать или дополнить его на уровне исходного кода или на уровне встраиваемых экранных скриптовых подпрограмм.

Оперирование данными на нескольких уровнях, с одной стороны, уменьшает защищенность системы от ошибок в программном коде, но с другой, позволяет создавать удобные, гибкие и быстрые инструменты обработки данных в соответствии с решаемой задачей. Так, например, рекурсивный расчет показателей связности производственного процесса на уровне DML сервера баз данных на порядок увеличил скорость выполнения этой процедуры, по сравнению с ее реализацией на уровне исходного кода приложения. С другой стороны, поддержка скриптовых подпрограмм позволяет обеспечить гибкость и простоту адаптации системы в процессе ее эксплуатации.

Поддержка скриптовых подпрограмм. В АСМА «Графит» интегрирован модуль выполнения скриптовых подпрограмм Fast Script. Данный инструмент был адаптирован для использования в модульной системе, а также расширен интерфейс работы с базой данных через объектное представление. Таким образом, в процессе эксплуатации возможна частичная модернизация и адаптация поведения системы под изменяющиеся требования, а

также новые требования, формируемые новыми технологическими процессами, но в пределах абстрактной модели данных.

Подсистема выполнения скриптов также имеет иерархическое представление:

- верхний уровень – глобальный юнит: функции и переменные, объявленные в данном юните, доступны из любой другой подпрограммы;
- уровень модуля – юнит, реализующий общие процедуры, обработчики для конкретного модуля, например, инициализацию начальных значений объектов, принадлежащих данному модулю;
- уровень экранной формы – юнит, подчиняющийся юниту модуля и реализующий обработчики для конкретной экранной формы, например, проверку разрешения редактирования объекта или определение набора видимых реквизитов;
- прикладной уровень – юнит, использующийся для расчета показателей прикладных объектов, например, количественных показателей партий продукции, карт операций.

Основными особенностями используемой реализации являются:

- стандартный языковой набор: переменные, константы, процедуры, функции (с возможностью вложенности) с переменными / постоянными / умалчиваемыми параметрами, все стандартные операторы и объявления (включая case, try/finally/except, with), типы (целый, дробный, логический, символьный, строковый, многомерные массивы, множество, variant), классы (с методами, событиями, свойствами, индексами и свойствами по умолчанию);
- проверка совместимости типов;
- доступ к любому объекту приложения;
- стандартные библиотеки для доступа к базовым классам, элементам управления, формам и БД;
- полное объектное описание структуры БД;
- широкий набор доступных для работы функций;
- поддержка работы с COM объектами;
- легко расширяемая архитектура.

Система версий программных модулей. В качестве поддержки версионности используется номер версии программных модулей, принятый в Windows-приложениях и библиотеках. Номер версии состоит из 4 цифр: версия продукта, версия модуля, релиз, сборка. Каждое число имеет 16-битовый формат, при совмещении – 64-битовый. Старшинство версии определяется сравнением 64-битовых значений. В информационной базе содержатся требования к версии программных модулей, а именно, к старшим 48 битам, так как младшие 16 бит определяют текущий номер сборки, которая не предполагает изменение функциональности.

Механизм обновления программных модулей. Обновление программных модулей должно производиться периодически и заключается в проверке появления новых версий и их загрузки из источника обновлений. Источник обновлений может быть трех типов:

- локальный или сетевой каталог;
- http ресурс с поддержкой работы через proxy-сервер;
- ftp ресурс с поддержкой работы через proxy-сервер.

В настройках программы пользователь может определить периодичность проверки обновлений. В ходе очередной проверки в случае присутствия обновлений модулей программы в источнике данных программа спросит пользователя о необходимости обновления. При подтверждении произойдет загрузка и автоматическая замена программных модулей, по окончании которой пользователь будет извещен о необходимости перезапуска программного комплекса.

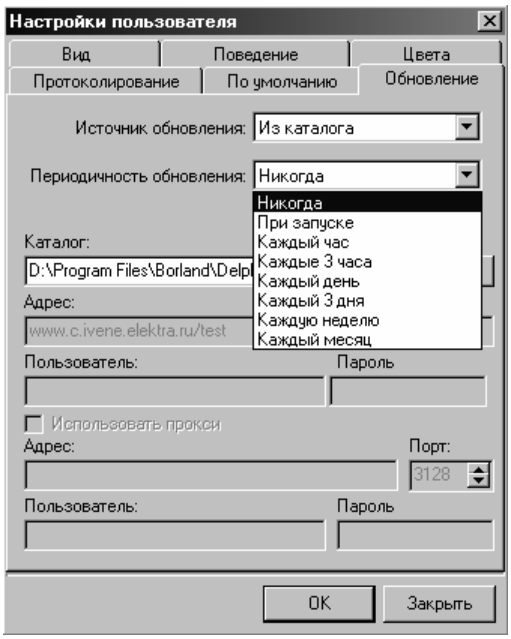

Рис. 5. Диалог настройки механизма обновлений программного комплекса.

Механизм репликации данных

Принятая модель распределенных баз данных. Используемая модель организации данных предполагает возможность развертывания до 4 баз с выполнением периодической репликации. В основу механизма разрешения конфликтов при репликации заложено разделение внутренних идентификаторов (ID) объектов на равные адресные пространства. Таким образом, программный пакет работает в общем пространстве внутренних идентификаторов базы, доступном ему на момент выполнения действий, а вновь создаваемые объекты находятся исключительно в его пространстве внутренних идентификаторов.

Алгоритм отслеживания изменений. В процессе работы с базой данных пользователи вносят изменения в содержание базы данных, изменения отслеживаются в протоколе работы. Таким образом, на основе данных из этой таблицы можно составить дельта-пакет для переноса информации в удаленную базу данных. На основе дополнительных сведений в протоколе работы о выполнении предыдущей репликации можно четко определить список измененных (удаленных, добавленных) объектов.

Репликация выполняется в 2 этапа. Так как в основу заложен принцип равенства баз данных, то не имеет значения, какая из сторон инициирует процесс репликации. Первым этапом будет сформирован дельта-пакет из локальной базы данных. После загрузки дельта-пакета удаленной базой данных будет сформирован ответный дельта-пакет для загрузки в локальной базе.

Разрешение конфликтов связанных с изменением одной и той же записи в 2х разных базах, решается вре-

менем выполнения изменений, за корректные принимаются более поздние изменения.

Разрешение конфликтов, связанных с изменением одной и той же записи в 2 разных базах, осуществляется во время выполнения изменений (за корректные принимаются более поздние изменения).

Выполнение репликации данных. Выполнение репликации данных начинается с выбора пункта «обмен данных» в меню сервис, данная функция доступна только в административном режиме. После выбора информационной базы производится выгрузка дельта-пакета, содержащего список изменений в базе дынных, произошедших от предыдущей синхронизации.

Дельта-пакет представляет собой архив в формате ZIP, который может быть передан с использованием произвольных средств доставки: дискеты, электронной почты, FTP и пр. При получении дельта-пакета по результатам загрузки будет сформирован ответный файл, который также следует передать в место инициации репликации, где он будет загружен. В результате базы данных будут иметь одинаковое наполнение.

Дополнительные средства формирования отчетности и интеграции со сторонними программными комплексами. Чтобы не ограничивать пользователя в выборе средств формирования выходных форм, разработан универсальный механизм формирования управленческой отчетности. Графически этот механизм представлен на рис. 6.

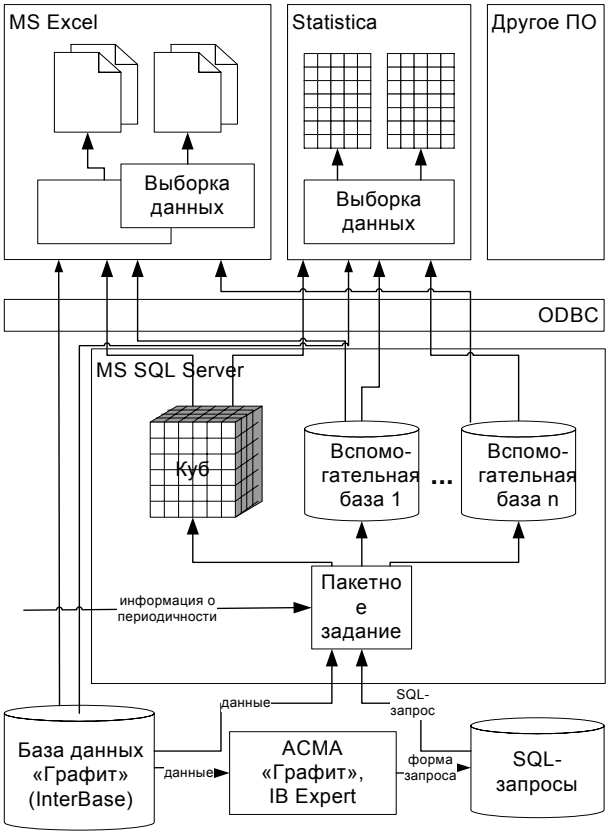

Рис. 6 Схема интеграции с внешними подсистемами.

К достоинствам данного механизма можно отнести то, что для извлечения данных используется язык построения запросов SQL 92, кроме того, существует возможность выгрузки сгенерированных отчетов в MS Excel для дальнейшего анализа.

Для облегчения процедуры формирования запросов разработан инструмент построения запросов SQL. На основе метаданных формируется дерево структуры БД. Пользователю представляется выбирать поля для отображения, а также задать условия отбора.

Методика формирования отчетности заключается в следующем:

1. В формировании запроса для извлечения данных из АСМА Графит.

2. Формировании промежуточных баз данных с возможностью задания интервала их актуализации, при необходимости долговременного хранения выгружаемых данных.

3. Загрузке данных в прикладную систему для формирования отчетности и выполнения операций анализа данных.

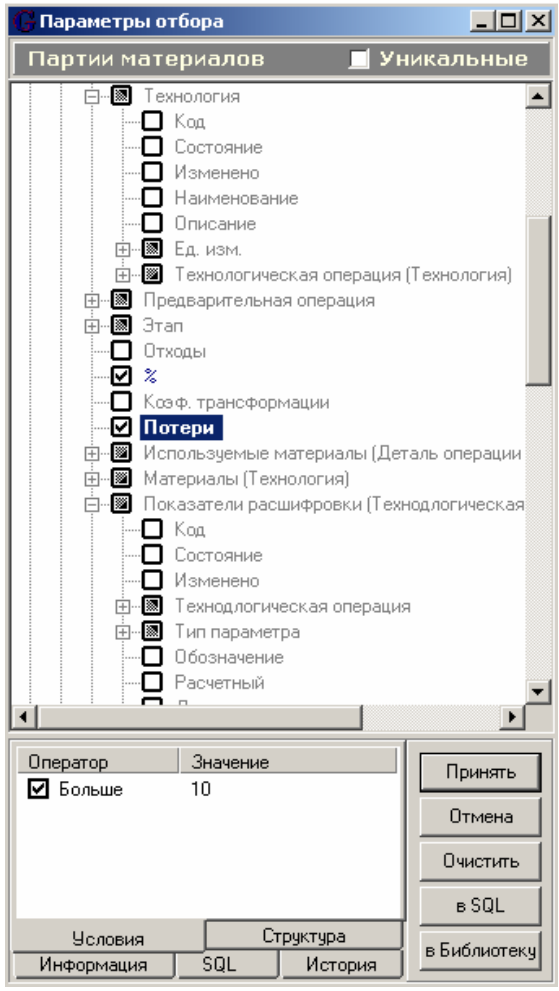

Рис. 7. Построитель запросов.

Использование типовых средств доступа к данным ODBC, интегрированных в семейство ОС Windows, позволяет абстрагироваться от конкретной СУБД, а также использовать продукты сторонних фирм для обработки и представления данных. Основными продуктами являются: MS Excel для формирования отчетных данных и выполнения простейшего анализа, входящий в пакет MS Office; Statistica фирмы StatSoft для аналитической обработки данных, поиска зависимостей, прогнозирования; Cristal Reports фирмы Seagate Software как мощный инструмент формирования отчетов.