Автоматизация проектирования прикладных информационных систем[1]

Вышинский Л.Л., Гринев И.Л., Флеров Ю.А., Широков А.Н., Широков Н.И.

Введение

Разработка прикладных программных систем – весьма трудоемкий и дорогостоящий процесс. Как правило, разработку таких систем ведут коллективы специалистов очень высокой квалификации. Основная сложность состоит в том, что эта работа требует, с одной стороны, определенного профессионального уровня в области компьютерных технологий, а с другой стороны, не менее профессионального уровня в понимании особенностей и тонкостей конкретной прикладной области. В настоящее время существует целый ряд подходов, так называемых CASE-технологий, которые предназначены для разработки прикладных программных проектов. RAD-технология, Vantage Team Builder, Power Builder, Oracle Designer и многие другие. Использование при разработке прикладных проектов подобных средств выглядит на первый взгляд заманчиво. Эти продукты более или менее интегрированы, разработаны и администрируются в едином стиле, обеспечивают достаточную функциональность для выполнения проекта. Однако существует и ряд отрицательных моментов при их использовании. Как правило, большинство этих продуктов

[1] Работа выполнена при частичной финансовой поддержке Минпромнауки (контракт
№ 37.011.11.0019) и комплексной программы научных исследований Президиума РАН «Математическое моделирование, интеллектуальные системы и управление нелинейными механическими системами» (проект 2.31)

предназначены для разработки прикладных информационных систем в универсальных интегрированных средах, которые позволяют решать максимально широкий класс задач. Однако ценой универсальности и интеграции прикладных проектов в эти среды является удорожание прикладной системы, большие объемы программного кода, понижение эффективности, наличие большого количества настроечных параметров и как следствие плохо управляемый процесс настройки системы, и наконец, большие затраты на эксплуатацию и модификацию системы. Для практического использования таких систем, приходиться их оснащать различными дополнительными программами, библиотеками, различными заготовками, приспособлениями и прочее. Создание этого программного и информационного окружения само по себе довольно сложно и хлопотно и поэтому их приобретение во многом теряет смысл – все равно пользователь не получает то, что ему нужно.

В противоположность использованию универсальных инструментальных сред многие разработчики исповедуют другой подход к разработке прикладных информационных систем, который построен на принципе “от задачи”. В этом случае зачастую фирмы, банки, предприятия принимают решение о разработке собственных программных реализаций для своих задач. Иногда они заказывают разработки профессиональным компьютерным компаниям, но часто прикладные программы разрабатывают сами для себя. Как правило, в этом случае разработчиками программ являются специалисты прикладных областей, которые с той или иной степенью успешности переквалифицировались в программистов. Надо заметить, что такой «самодеятельный» подход к разработке программных систем обладает определенными достоинствами. Это оперативность, управляемость, целенаправленность разработки, относительно низкая стоимость. Однако эти достоинства часто оборачиваются в свою противоположность. «Оперативность» оборачивается непродуманностью общей концепции и, как следствие, изъянами в моделях и структурах данных. «Управляемость» приводит к фактически неуправляемому введению в систему постоянных «улучшений», исправлений, модификаций, многочисленных

«заплат», а в итоге кажущаяся низкая стоимость «домашней» разработки оборачивается регулярной данью, которую надо платить за поддержание системы в мало-мальски рабочем состоянии. Это особенно проявляется тогда, когда речь идет о комплексной автоматизации таких сложных технологических процессов, как проектирование, управление производством, управление крупным торговым предприятием, банком и тому подобное. Причиной тому является отсутствие технологии разработки прикладных информационных систем и удобных, доступных средств, которые позволяли бы полностью или частично устранить разрыв между предметной областью и реализацией необходимых задач в виде эффективно работающего программного обеспечения. Настоящая статья как раз и посвящена обсуждению этих вопросов, а также изложению авторского подхода к формализации и автоматизации проектирования, разработки и сопровождения сложных прикладных информационных систем.

Проектный подход к разработке информационных систем

Вообще говоря, всегда между постановкой проблемы и ее реализацией существует разрыв. Но в большинстве сфер производственной деятельности его научились преодолевать за счет квази – реализации, то есть за счет создания **проектов**. Это непреложная истина: для того, чтобы избежать поспешных решений и неудачных реализаций, для того чтобы получить продуманную, добротную и красивую вещь (здание, конструкцию, машину, программную систему) необходимо сначала разработать ее проект, со всех сторон проанализировать его, обсудить со специалистами, может быть построить макет, и лишь после этого приступать к реализации, неукоснительно следуя проекту. Во многих традиционных сферах этап проектирования узаконен и существуют определенные правила и стандарты создания проектов. Но этого мало, между проектом и конечным объектом, как правило, стоит технология, которая определяет необходимое оборудование, инструменты, оснастку и другие средства конкретной реализации

проекта. В области прикладного программирования ситуация несколько иная. Конечно, любой программной разработке обязательно предшествует некий план или техническое задание, или даже проект, например, в виде набора квадратиков и стрелочек между ними. Существуют даже определенные подходы к построению таких проектов. Одним из наиболее развитых методов проектирования информационных систем является структурный анализ и функциональное моделирование, которые были положены в основу концепции SADT - Structured Analysis and Design Technique ([1,2]) и которая появилась еще в семидесятые годы. На основе SADT в ряде отраслей были даже разработаны стандарты проектирования больших информационных систем ([3]). Как показал опыт, использование таких подходов весьма полезно для этапа исследования предметной области, построения структурной модели системы автоматизации этой области, ее функциональной полноты и непротиворечивости. Однако построение структурной модели не исчерпывает все содержательные проблемы, которые хотелось бы решать на этапе проектирования. Проект системы, с нашей точки зрения, с точки зрения программистов, которые его реализуют, должен содержать полную информацию о том, с какими содержательными понятиями и объектами, с какими математическими моделями в данной системе будут иметь дело пользователи, как эти объекты связанны между собой, каковы свойства этих объектов, что с этими объектами можно будет делать, какие функции и процедуры должны быть реализованы в системе, в каком виде должны быть представлены результаты и так далее и тому подобное. Только такое максимально детализированное описание задачи может в полной степени соответствовать понятию проекта и служить отправной точкой для программирования. Здесь могут задать вопрос, а чем, собственно, полный текст программы на каком-нибудь алгоритмическом языке программирования плох в качестве проекта. Ведь в этом тексте однозначно определены все содержательные понятия и модели? Это действительно так, но в тексте программного кода очень трудно, а практически невозможно, отделить содержательные вопросы от чисто программистских аспектов, связанных с реализацией проекта в конкретной

вычислительной среде. А такое разделение, на наш взгляд, крайне полезно как с точки зрения специализации разработчиков, так и для более четкого выделения и понимания всех, возникающих проблем, и содержательных, и программистских.

С содержательной стороны составление программного кода является технической стороной дела, однако, это не означает, что этот аспект разработки прост и, дескать, не следует о нем беспокоиться. Как раз наоборот. Программирование это очень сложный, один из наиболее наукоемких технологических процессов. Действительно, программирование требует специальных знаний в области современных методов и языков, знания основных возможностей операционных систем, систем управления базами данных, графических средств отображения, средств телекоммуникаций и тому подобное. Для того, чтобы уровень программной реализации был бы достаточно высок, одного профессионализма программистов не достаточно - нужны еще хорошо развитые инструментальные средства, которые бы в наибольшей степени соответствовали разрабатываемой системе. Идеальным, с нашей точки зрения, является такой инструментарий, который полностью автоматизирует, то есть, генерирует создание программного кода по проекту системы. В этом случае задачей программиста является разработка такого инструментария или, если необходимо, настройка его под конкретные особенности разрабатываемой системы. Это как раз и означает, что программист становится технологом в полном смысле этого слова.

Надо сказать, что наши взгляды на создание программных проектов формировались постепенно, в процессе более чем двадцатилетней практики разработки прикладных информационных систем. На этом пути нами была создана серия различных технологических инструментальных средств [4,5,6,7], которые в течение долгого времени позволяли относительно небольшому коллективу разрабатывать, внедрять и сопровождать различные прикладные информационные системы в самых разных предметных областях.

Согласно нашему пониманию проектного подхода, весь процесс разработки прикладных систем состоит из двух этапов. Первый этап - это собственно проектирование системы. Проект прикладной программной системы с нашей точки зрения – это формальный документ или совокупность формальных документов, которые обладают определенной структурой, определенным составом своих компонент, правилами оформления, синтаксисом, семантикой и прочими атрибутами формального объекта. Проект должен адекватно отражать ту предметную область, для которой разрабатывается система. Проект должен быть не только понятен специалистам прикладникам, но эти специалисты, как правило, не искушенные в программировании, должны иметь возможность непосредственно участвовать в проектировании системы. Естественно, что для этого необходимы общедоступные правила построения проектов или отдельных его компонент, соответствующие формальные языковые и программные средства. Здесь могут быть весьма полезна и методика SADT или другие аналогичные подходы к проектированию. Однако существенное требование в проектном подходе состоит в том, чтобы уровень описания и формализации проекта был бы достаточным для однозначного истолкования его на следующем технологическом этапе, на этапе генерации программного кода. Средства описания различных понятий, объектов и функций, необходимых в проекте, могут быть разными для разных компонент проекта. Очевидно, что должны быть средства самого общего описания проекта, определяющие его идентификационные реквизиты, такие как наименование, дату начала проекта, дату и номер последней версии, исполнителей, состав основных программных компонент и так далее. Для описания основных содержательных понятий и объектов предметной области могут использоваться специальные непроцедурные языки высокого уровня. Для описания интерфейсов должны быть свои языки, определяющие форматы передачи информации между программными компонентами. Очевидно, что наилучшим способом описания некоторых функций, необходимых при решении той или иной задачи, является просто программный текст на универсальном алгоритмическом языке или ссылка на

определенное имя в библиотеке программ. Естественно, что совершенно исключать возможность появления программного кода в исходном проекте было бы неразумно. Важно только, чтобы эти вставки были хорошо специфицированы, чтобы они могли быть корректно включены в результирующий программный код системы.

Второй этап – это генерация полного программного кода системы и его технологическая сборка, то есть создание инсталляционного пакета. Второй этап должен быть полностью автоматизирован и не должен допускать изменений с содержательной стороны. Этот этап отражает технологию разработки проекта. Автоматизация требует специального инструментария, который является технологической компонентой проекта. Фактически, инструментарий – это генератор программного кода проекта (**ГЕНЕРАТОР ПРОЕКТА**). С нашей точки зрения, генератор проекта является неотъемлемой компонентой всей системы в целом, причем, не только на стадии разработки, но и в течение всего ее жизненного цикла. Жизнь проекта программной системы не заканчивается ее реализацией. Длительность жизненного цикла программных систем определяется способностью к модификациям. Как показывает опыт, модификации неизбежны для живущих систем. Изменения условий и правил внешнего мира, устаревание используемого оборудования, новые более совершенные общесистемные средства – все это стимулирует модификацию действующей системы. Однако исправления работающих систем является весьма дорогим и сложным процессом. Если же при разработке системы использовался проектный подход, то процедура модификации существенно упрощается за счет того, что изменения вносятся только в проект, а все дальнейшие изменения в программном коде, в информационном окружении и в прочих компонентах проекта должны осуществляться автоматически. В связи с этим, проектный подход имеет смысл не только на этапе разработки системы, но и в процессе ее эксплуатации и сопровождения.

Разумеется, для разных типов прикладных систем нужен разный инструментарий. Одно дело если разрабатывается

автономная локальная программа, рассчитанная на одного пользователя, и совершенно другое, если речь идет о разработке многопользовательской системы. Одно дело если в системе требуется реализовать сложные длительные расчеты и другое дело, если нужно просто собирать и хранить информацию из десятков и сотен различных источников. Проектный подход не подразумевает применение некого единого инструментария, некоторого универсального «ГЕНЕРАТОРА», годящегося на все случаи жизни. Скорее всего, для каждого класса задач в рамках проектного подхода нужно специально разрабатывать свой «генератор». Мы здесь хотим продемонстрировать преимущество проектного подхода на классе многопользовательских информационно поисковых систем, которые укладываются в архитектуру типа «клиент – сервер».

Архитектура «клиент – сервер»

Обычно под термином «клиент-сервер» понимают архитектуру многопользовательских систем, которая предусматривает наличие клиентских и серверных программных компонент. Клиентские модули используются на удаленных рабочих местах пользователей, а централизованные серверные программы обеспечивают «обслуживание клиентов», то есть прием удаленных запросов пользователей, их обработку и возврат им же результатов этой обработки. Примером клиент - серверных систем являются средства обработки удаленных SQL запросов к реляционным базам данных в некоторых интегрированных СУБД. Это простейшая двухуровневая архитектура «клиент – сервер» изображена на рис. 1.

Рис. 1. Двухуровневый «клиент – сервер»

Клиентский модуль (АРМ пользователя) и СУБД могут находиться как на одном компьютере, так и на разных. В последнем

случае связь между ними осуществляется по локальным или публичным телекоммуникационным каналам. Однако в этой архитектуре прикладных систем, при одновременной работе нескольких пользователей, которые выполняют сложные запросы с модификацией данных, иногда может возникать потеря целостности данных или некорректное их использование.

Классической же структурой прикладных многопользовательских информационно – поисковых систем является архитектура «трехуровневый клиент-сервер». Разрабатываемые в рамках этой архитектуры программные системы имеют уже три уровня программных компонент:

- клиентские модули;
- прикладной программный сервер системы;
- система управления базой данных.

Преимуществом трехуровневой архитектуры над двухуровневой является то, что в них режим доступа к данным регулируется прикладным программным сервером. В частности, может быть реализован режим последовательной обработки клиентских запросов. Последовательная обработка запросов пользователей – это одна из важнейших черт клиент – серверной архитектуры. В значительной степени благодаря этому свойству решается проблема поддержания целостности данных в системе.

На рисунке 2. показана схема взаимодействия программных компонент в трехуровневой структуре клиент - сервер.

Рис. 2. Трехуровневый «клиент – сервер»

Клиентский модуль (АРМ пользователя) системы взаимодействует с прикладным сервером, разработанным специально для данного приложения. Поэтому форма клиентских запросов здесь не ограничивается языком SQL. Прикладной сервер

на основе информации хранящейся в базе данных может обеспечить выполнение любого сложного анализа поступающих запросов, в том числе и проверок полномочий пользователей.

Прикладной сервер системы взаимодействует с сервером базы данных, используя стандартный интерфейс, поддерживаемый применяемой СУБД. При этом клиентские компьютеры при правильной организации сети не имеют непосредственного доступа к таблицам базы данных. С точки зрения системы управления базой данных ее клиентом является прикладной сервер системы, а не клиентские модули. Программные компоненты приведенных трех уровней могут размещаться на разных компьютерах и взаимодействовать между собой по телекоммуникационным каналам.

Однако возможны и более сложные схемы взаимодействия пользователей с серверными программами, которые связаны либо с обеспечением информационной безопасности, либо работой через ИНТЕРНЕТ, либо работой с распределенными данными и так далее. Поэтому возникает необходимость рассматривать более сложную архитектуру клиент – серверных систем, которую принято называть «многоуровневый клиент – сервер».

Структурная модель проектируемых систем

Общая архитектура проектируемых систем. В дальнейшем речь пойдет о проектном подходе и об автоматической генерации программного кода в рамках архитектуры «многоуровневый клиент – сервер». Для более точного и полного определения этого класса проектируемых систем представим его в виде структурной модели. Под структурной моделью мы здесь понимаем состав программных компонент системы и связи этих компонент между собой. В общем случае структурная модель проектируемых систем в рамках описываемого здесь проектного подхода представлена на рисунке 3. Информационные системы, имеющие такую структуру, могут решать самый широкий класс прикладных задач.

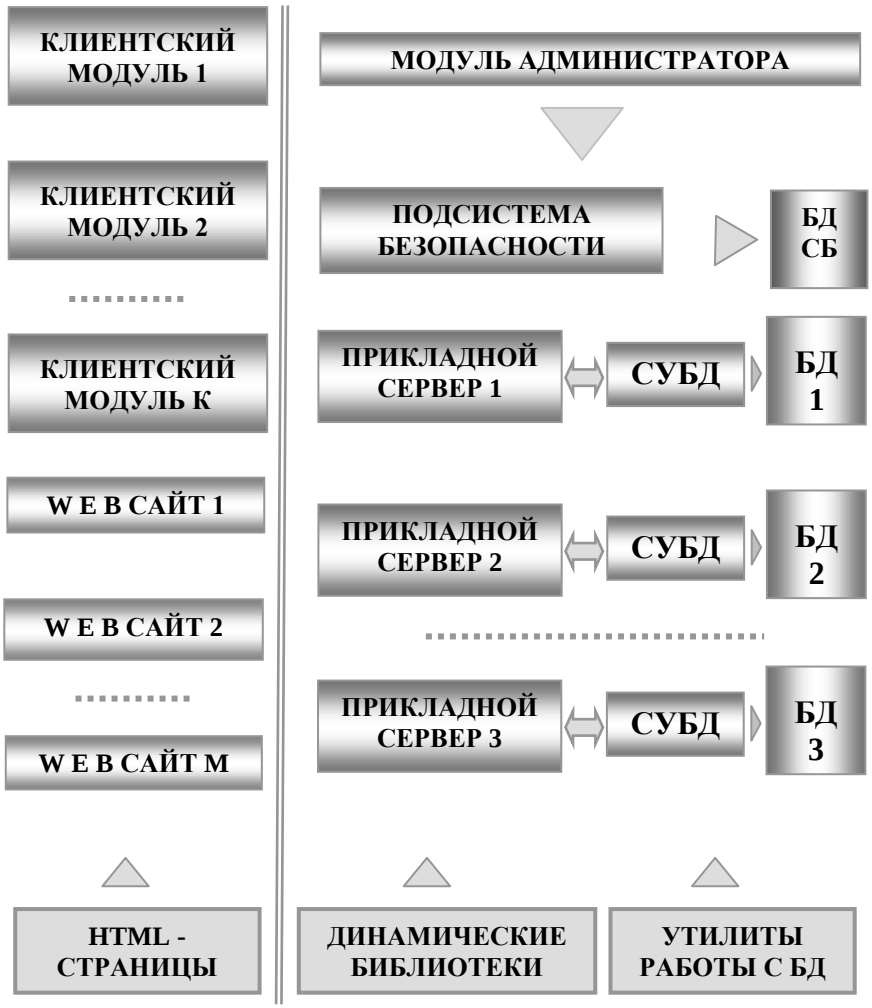

Рис. 3. Структурная схема систем, проектируемых с помощью ГЕНЕРАТОРА ПРОЕКТОВ

Из приведенной структурной модели видно, что в общем случае это многопользовательские системы с распределенным способом хранения данных. Приведенные на рисунке программные компоненты могут размещаться на разных, в том числе и удаленных вычислительных комплексах, впрочем, некоторые из них могут находиться и на одном компьютере, если это не противоречит логике работы конкретной системы. Все программные компоненты этой структуры информационно связаны между собой, а технически эта связь может осуществляться по локальным, корпоративным или публичным телекоммуникационным каналам, в том числе и по каналам ИНТЕРНЕТ.

Клиенты. Пользователи таких систем реализуют свои функции с помощью специальных программных компонент системы – клиентских модулей (**КМ**). Клиентские модули могут быть разных типов, в зависимости от функций различных пользователей или групп пользователей. Количество экземпляров клиентских модулей одного типа ограничиваются только спецификой этого типа и логикой работы системы. Разные экземпляры клиентских модулей работают независимо друг от друга, но могут обмениваться информацией между собой посредством обращения к общим данным, хранящимся в системе. Это не исключает передачу между пользователями адресных сообщений в определенном прикладной системой формате.

Основной задачей клиентских модулей является инициация выполнения функций системы. В системах типа клиент - сервер инициатива действия всегда принадлежит клиентской стороне. Для выполнения функций на клиентском модуле должны быть подготовлены все необходимые данные для конкретного действия и переданы от клиента одному из доступных прикладных серверов, с указанием кода (идентификатора, номера) команды. Прикладной сервер (**ПС**), получивший команду от клиента, выполняет предусмотренные этой командой действия и возвращает на клиентский модуль информацию, являющуюся результатом выполнения команды. Таким образом, все действия систем рассматриваемого типа имеют следующую схему:

<**КМ**> → <входные данные> → <**ПС**> → <результат> → <**КМ**>

Способы подготовки входной информации пользователями и характер использования результатов выполнения команды зависят от категории пользователей, на которых ориентированы те или функции системы. Приведенную цепочку действий, включая способ подготовки входной информации на клиентской стороне и действия, связанные с обработкой клиентским модулем выходной информации, будем называть пользовательской процедурой (или просто процедурой). Часть пользовательской процедуры, связанную с обращением к прикладному серверу

<входные данные> → <**ПС**> → <результат >

будем называть запросом к прикладному серверу (или просто запросом). Характер или тип запроса определяет код команды, который является обязательной частью входных данных. Формат входных и выходных данных определяется типом запроса. Очевидно, что запросы инвариантны относительно способов подготовки входной информации и обработки выходной. Поэтому один и тот же запрос может быть использован в разных процедурах и в разных типах клиентских модулей.

Многие современные системы коллективного пользования предоставляют возможность работать большим группам своих клиентов через ИНТЕРНЕТ. В рамках представленной структурной модели для этой цели служат специализированные WEB-серверы. Используя только стандартный ИНТЕРНЕТ-броузер, пользователи в рамках выделенных им полномочий могут выполнять определенные запросы к прикладным серверам. WEB – серверы по сути являются специальными клиентскими модулями коллективного пользования.

Часто логика функционирования прикладных информационных систем, их место во внешней среде, требует обеспечения интерфейса с другими программными системами. Для этой цели в нашей структурной модели предусмотрена возможность формирования различных динамических библиотек, которые содержат функции, реализующие запросы к прикладным серверам.

Серверы. Каждый запрос адресуется одному из прикладных серверов системы. Однако при необходимости прикладной сервер сам может обращаться с запросами к другим прикладным серверам. То есть, прикладные серверы между собой тоже могут находиться в отношении «клиент-сервер». Как правило, в прикладных информационных системах запросы связаны с поиском, хранением и модификацией информации в базах данных. В рамках рассматриваемой структурной схемы каждый прикладной сервер непосредственно может быть связан не более чем с одной базой данных. Непосредственная связь с базой данных осуществляется средствами интерфейса, которые предоставляет СУБД. И хотя непосредственная связь в рамках нашей модели возможна только с одной базой данных, используя запросы к другим серверам, можно одновременно работать разными базами.

В принципе разные серверы могут работать с одной и той же базой данных, однако в этом случае необходимо не допускать пересечение запросов от разных серверов и предпринимать специальные меры для поддерживания целостности данных.

Взаимодействие клиентских модулей с прикладными серверами в нашей модели осуществляется посредством транспортного сетевого протокола TCP/IP. Этот протокол является в настоящее время наиболее популярным и позволяет организовать работу системы в локальной сети предприятия, в глобальной корпоративной сети или в глобальной сети ИНЕРНЕТ. На содержательном уровне протокол общения между клиентским модулем и сервером должен носить документарный характер, то есть, каждая содержательная порция данных, передаваемая от клиента к серверу и обратно, должна иметь формат одного из документов, которые определены в рамках прикладной системы.

Система безопасности. В силу того, что доступ к информационным ресурсам сервера может осуществляться через публичные каналы связи, особое внимание в нашей модели уделено проблемам безопасности. Функции системы безопасности, которая исключительно для наглядности выделена на рисунке 3 в отдельный блок, в реальности распределены по всем компонентам системы. В

системе безопасности решаются две задачи. Первая – это аутентификация пользователей и проверка их полномочий доступа к ресурсам серверов, а вторая - это защита информации при передаче по телекоммуникационным каналам.

Для установления взаимной подлинности клиента и сервера в системе могут применяться различные способы аутентификации. Аутентификация на симметричных ключах подразумевает ведение базы данных пользователей с соответствующими им паролями. Эта база данных хранится на серверной стороне. На клиентской стороне вводится имя и пароль при каждом подключении к серверу. Это простейший способ аутентификации. Аутентификация с применением, так называемых, сертификатов использует несимметричные ключи, состоящие из двух частей: закрытый ключ, хранящийся на персональном носителе (дискета, таблетка, смарт-карта), защищенном паролем, и открытый ключ, помещенный в сертификат, и доступный широкому кругу пользователей. Общая концепция подсистемы безопасности с применением сертификатов подразумевает построение иерархической структуры доверительных отношений. Для этого создается корневой сертификационный центр, сертификат которого помещается в доверяемые базы данных всех пользователей системы. Этот центр выпускает сертификаты для подчиненных сертификационных центров. В нашем случае должен быть создан сертификационный центр системы, который либо является корневым, либо входит в иерархию сертификационных центров той среды, в которой система должна работать. На уровне сертификационного центра системы выдаются сертификаты администраторам нижнего уровня, администраторам отдельных серверов. На уровне отдельного сервера системы выдаются сертификаты своим клиентам. Перекрестные связи между серверами обеспечиваются размещением сертификатов администраторов этих серверов в соответствующих доверительных базах данных. Такая конфигурация системы безопасности имеет свое преимущество, поскольку позволяет пользователям различных систем, с одним корневым сертификатом, работать с серверами любой из систем этой группы без изменения списка доверяемых сертификатов.

Модуль администратора входит в систему безопасности и обеспечивает ведение базы данных пользователей системы их паролей, ключей, сертификатов, а также функциональное разграничение полномочий различных групп пользователей. Надо сказать, что для относительно небольших систем допустимо совмещение полномочий администраторов разного уровня в одном лице. Например, администратор системы и администратор сервера может быть одним лицом.

Для защиты информационного канала сервера от несанкционированного доступа, т.е. от просмотра и модификации информации посторонними лицами, используются различные методы шифрования/дешифрования данных в каналах передачи. Вся информация, которая передается между клиентом и сервером после аутентификации, шифруется с применением симметричных сессионных ключей. Выработанные случайным образом в процессе аутентификации сессионные ключи действуют только на период текущей сессии. Сессия в подсистеме безопасности начинается с момента установления сетевого соединения с клиентом и заканчивается после разрыва соединения с клиентом.

Утилиты по работе с базами данных. Важным аспектом разработки прикладных систем является разработка специальных средств для инсталляции и сопровождения системы. Для поддержания этих функций в нашей структурной модели предусматривается специальная утилита - конфигуратор баз данных. Конфигуратор баз данных используется для построения командных файлов создания пустой базы данных, создания основных объектов базы данных (таблиц, индексов), удаление их из базы данных, а также обновление баз данных при модификации системы.

В конфигураторе базы данных предусмотрена также функция генерации командных файлов загрузки и выгрузки информации в текстовые файлы. Такая загрузка и выгрузка может использоваться как для резервного копирования наряду со штатными средствами системы управления базой данных, так и для переноса информации в базу данных под управлением другой СУБД. Например, можно выгрузить информацию из базы данных

MS SQL Server и загрузить в базу данных Oracle, используя промежуточное представление в виде текста.

Многоплатформенность системы. Конфигурация баз данных является частным случаем конфигурации аппаратно-программной платформы, на которой должна работать прикладная система. В условиях стремительного прогресса, как аппаратных средств компьютерных систем, так и программного обеспечения, актуальной становится задача разработки таких проектов, которые могут работать на различных платформах. Платформы могут отличаться аппаратурой, операционной системой, системами управления базами данных. При этом разные компоненты системы должны уметь работать на разных компьютерах и под разными платформами. Требование многоплатформенности подразумевает сохранение неизменным интерфейса между клиентскими модулями и серверами системы. Это означает, что один и тот же клиентский модуль одинаково успешно работает с серверами на разных платформах, а разные исполнения клиентских модулей на различных платформах могут работать с одним и тем же сервером. Такое свойство обеспечивается стандартизацией протокола взаимодействия клиента и сервера. Другое проявление многоплатформенности - это одинаково успешная работа одного сервера с различными типами систем управления базой данных или с различными интерфейсами.

Решение задачи многоплатформенности в нашем подходе к проектированию систем отнесено к технологическим проблемам. Решаются эти проблемы на этапе генерации программного кода, указанием конкретных платформ под которые необходимо генерировать систему или ее отдельные компоненты, а также генерации в составе системы специальных конфигурационных файлов. При изменении конфигурации платформы всегда можно перенастроить или даже перегенерировать систему без каких-либо изменений в исходном проекте.

Проект системы

Структура проекта. Поскольку проект представляет собой формальный объект, то, прежде всего, должны быть определены среда, в которой он существует, правила его представления и оформления, а также структура проекта, то есть состав, связи и свойства основных его составляющих. В рамках нашей технологии под проектом системы мы понимаем совокупность файлов, организованную в виде отдельной директории, имя которой совпадает с именем проекта. В структуре этой директории можно выделить следующие каталоги, в которых хранятся:

- текущая версия генератора проекта;
- полный комплект файлов описания проекта;
- полный комплект файлов генерируемого программного кода системы;
- полный комплект исполняемых файлов системы.

Как говорилось выше, в проектном подходе ГЕНЕРАТОР является неотъемлемой частью системы. Действительно, программный код является функцией проекта и инструментальных средств, с помощью которых он создается. Поэтому очень важно, чтобы текущая версия генератора сохранялась в течение всего жизненного цикла прикладной системы. Необходимо постоянно поддерживать соответствие программного кода текущей версии генератора.

Каталог проекта содержит полное его описание, которое состоит из файлов следующих типов:

- <имя проекта>.gen – головной файл проекта. Этот файл содержит общие реквизиты проекта, списки проектных констант, ряд других общепроектных объектов, а также перечень компонент проектируемой системы, т.е. клиентские модули, WEB-серверы, прикладные серверы, библиотеки и прочее;

- <имя проекта>.typ – файл, содержащий описание типов и структур данных системы;
- *.srv – файлы описания прикладных серверов, которые содержат описание баз данных и запросов к данному серверу;
- *.mdl – файлы описания клиентских модулей системы, то есть описание допустимых пользовательских процедур;
- *.sit – файлы описания HTML – страниц и способов обращения с этих страниц к прикладным серверам;
- *.c, ..*.h – файлы, которые содержат программный код для реализации некоторых запросов, которые в проекте специфицированы как объекты программируемые «вручную», в отличие от автоматически создаваемых генератором;
- icons\ *.ico – файлы, представляющие собой пиктограммы, использующиеся в проекте. Эти файлы сведены в отдельный каталог, принадлежащий проекту (project \ icons).

Остановимся подробно на основных компонентах проекта. Однако в дальнейшем описание проекта для наглядности мы продемонстрируем на небольшом примере.

Головной файл проекта. Головной файл содержит ряд разделов. Эти разделы в файле не выделяются заголовками, но их порядок строго определен и контролируется по начальным ключевым словам:

1. Заголовок проекта.
2. Параметры подсистемы безопасности.
3. Список используемых в проекте пиктограмм.
4. Список используемых в проекте файлов изображений (для WEB-интерфейса).
5. Список поддиректорий сервера, доступных из программного кода.

6. Список кодов ошибок с текстами диагностики..
7. Список констант.
8. Список спецификаций запросов клиентских модулей к прикладным серверам других проектов.
9. Список прикладных серверов проекта.
10. Список спецификаций запросов данного проекта для использования в других проектах.
11. Список клиентских модулей.
12. Список WEB-сайтов.
13. Список спецификаций динамических библиотек, используемых модулями системы.
14. Перечень «ручных» программных файлов, которые необходимо включить в проект.

Рассмотрим наиболее важные из этих разделов.

Заголовок проекта. В заголовке проекта задается идентификатор проекта, полное наименование и несколько опций:
project <имя проекта> : <строка комментария>;
<опции проекта>

Опции задают номер версии, копирайт, начальный номер TCP/IP портов для распределения по серверам, список требуемых интерфейсных драйверов баз данных, возможно имена авторов проекта и пр. Отметим отдельно значение опции, определяющей взаимодействие с базами данных. В составе системы могут быть реализованы различные драйверы работы с базами данных. В частности, драйверы для работы через ODBC (с произвольной СУБД), через библиотеку ntwdblib MS SQL Server 6.5/7.0/2000 в MS Windows NT/2000, через библиотеку ociw32 Oracle v7.0/8.0 в MS Windows NT/2000, через библиотеку psql Postgesql в ОС Linux, через библиотеку ctlib Sybase в MS Windows NT/2000 и ОС Linux. Приведенный список может быть легко расширен. Список конкретных драйверов, использующихся в проекте указывается в специальной опции */database=(<список драйверов>).*

Список прикладных серверов проекта. Здесь перечисляются все прикладные серверы текущего проекта:

server <имя сервера>; <опции сервера>...

Для каждого перечисленного здесь сервера в составе проекта должен быть файл описания сервера - <*имя сервера*>.srv.

Список клиентских модулей. В этом разделе задается список клиентских модулей проекта.

module < имя модуля > :<строка комментарий>; <опции модуля>...

Для каждого перечисленного здесь модуля в составе проекта должен быть файл описания модуля < *имя модуля* >.mdl.

Список WEB-сайтов. WEB-сайты, как уже говорилось выше, являются специфическими клиентскими модулями, для обеспечения работы через ИНТЕРНЕТ. В этом разделе задается список WEB-сайтов проекта:

site < имя сайта > :<строка комментарий>; <опции сайта>…

Список спецификаций запросов данного проекта для использования в других проектах. Любые запросы проектируемой системы могут быть использованы в других проектах. То есть, проектируемая система открыта для внешнего использования. Однако для этого нужна санкция, заложенная в проект системы. В данном разделе перечисляются имена некоторой совокупности запросов, которые становятся доступными для других проектов или для программ, написанных без использования генератора. Для каждой такой именованной группы запросов генерируется соответствующий файл спецификаций. Принадлежность конкретных запросов некоторой группе задается при описании самого запроса. Поэтому необходимо определить группу, в которую в дальнейшем поместим ряд запросов, открытых для других систем.

export <имя запроса>; <опции экспорта>…

Здесь опции определяют форму использования запросов (cgi, library) Наличие таких опций обусловливает генерацию соответствующих программных компонент. Отсутствие опций рассматривается как использование запросов в других проектах в качестве импорта.

Список спецификаций запросов клиентских модулей к прикладным серверам других проектов. Так же как проектируемая система открыта для внешнего использования, так и в самом проекте могут быть использованы запросы к прикладным серверам других систем. Для этого из каждого такого проекта берется файл спецификаций именованной совокупности запросов. В данном разделе задается имя проекта и имя совокупности спецификаций. В случае возникновения конфликтов имен типов данных здесь может быть задан список переименований этих имен. Имена проектов, совокупностей запросов и самих запросов не переименовываются и должны быть уникальными.

Параметры подсистемы безопасности. В этом разделе задается связь специфических понятий подсистемы безопасности с типами данных, декларированными в проекте (о типах данных будет сказано ниже). Например, задаются типы данных, которые будут использоваться для представления имен пользователей, паролей, сертификатов и пр.

Перечень программных файлов включаемых в проект. Проектный подход полностью не исключает возможность ручного программирования отдельных функций. Мы просто стремимся уменьшить его объем за счет автоматизации создания объемного программного кода для многих функций, которые допускают либо стандартизацию при программировании, либо эффективную формализацию для описания в проекте. В этом разделе задается список файлов с программным кодом, написанных вне проекта, и необходимых для включения в его состав.

Описание типов данных. Все константы, переменные и параметры, встречающиеся в описании проекта, должны быть отнесены к одному из представленных в файле описания типов данных. Данные могут быть простыми или структурированными, то есть имеющими некоторые явно выделенные и поименованные компоненты. Описание типов данных должно содержать идентификатор типа, указания на его внутреннее и внешнее представление, состав его компонент, а также название и обозначение:

type <идентификатор>:
{<формат>(<базовый тип>)|alias <идентификатор типа>}
[(<список значений>|<список свойств>)]
/tilte=<строка комментария>/hdr=<строка заголовка>

Внутреннее представление должно соответствовать одному из базовых типов данных: int, short, double, char. Для символьных данных в квадратных скобках указывается размер строки: (char[число]). Внешнее представление данных при вводе и выводе определяется идентификатором формата, который применяется для этого типа данных. Каждому базовому типу соответствует несколько возможных форматов представления. Формат затрагивает не только внешнее представление, но и множество допустимых значений описываемого типа данных. Естественно полагать, что если значение какой-то переменной не соответствует внешнему представлению, то есть, не может быть представлено в описанном формате или это представление не будет иметь практического смысла, то это значение недопустимо для данного типа.

Тип данных	Формат	Комментарий
int	numb enum combo radio mask date ltime	целое число (по умолчанию), перечислимый тип, перечислимый тип (в диалогах в виде combo box), перечислимый тип (в диалогах в виде группы radio button), бинарная маска (в диалогах в виде группы check box) , дата в виде ДД/ММ/ГГГГ или ДД/ММ/ГГ, время с точностью до секунд в виде ЧЧ:ММ:СС
short	numb time	целое число (по умолчанию), время с точностью до минут в виде ЧЧ:ММ

double	numb money money3	десятичное число с точкой (по умолчанию), деньги в формате *.dd или *.ddd), тоже что и money, но с разрядкой по триадам целой части
char	char alpha	строка (по умолчанию), идентификатор

Представленные выше типы данных и их форматы составляют базовый набор. Этот набор может быть расширен или изменен при проектировании или конфигурировании конкретных прикладных систем. Конфигурирование проекта под конкретные условия применения системы может коснуться представления календарных дат, времени, денег и других общеупотребительных типов данных, которые могут зависеть от языка, на который ориентирован проект или от других специальных установок. Так, например, даты могут представляться не так как выше, а на американский манер в формате ММ/ДД/ГГГГ, или ММ-ДД-ГГГГ, или ММ.ДД.ГГГГ и так далее.

Среди перечисленных выше некоторые типы данных требуют структурированного представления. Для этого определены специальные форматы внешнего представления: enum, combo, radio, mask. Собственно эти типы данных не образуют во внутреннем представлении структуры (это просто целое число), но во внешнем представлении они имеют вид конечного упорядоченного списка возможных значений, снабженных текстовыми строками в качестве комментария. Значение перечислимых типов (enum, combo, radio) соответствует номеру выбранной строки в этом списке. Полное же определение перечислимого типа задается упорядоченным списком возможных значений:

<список значений> ::=
<идентификатор значения> / title = <строка комментария >
[,<идентификатор значения> / title = <строка комментария > ...]

Явно структурированным типом данных является целое число (int), представленное в виде бинарной маски : mask. Смысл этого представления состоит в том, что каждый бит двоичного представления целого числа, соответствует определенному свойству

(или признаку) из n заданных свойств (n<=32). Таким образом, произвольное целое число, не превосходящее 2^n-1, представляет некоторое подмножество декларируемых свойств. Описание представления целого числа в виде битовой маски задается полным списком декларируемых свойств.

<список свойств> ::=
<идентификатор свойства> / title = <строка комментария>
[,идентификатор свойства> / title = <строка комментария>…]

Описание прикладных серверов. Каждому прикладному серверу, упомянутому в файле описания проекта, соответствует отдельный srv файл описания сервера. Этот файл содержит следующие разделы.

1. Заголовок сервера.
2. Перечень описаний таблиц базы данных сервера.
3. Перечень индексов базы данных.
4. Перечень связей между таблицами базы данных.
5. Перечень описаний SQL-запросов, используемых в программах сервера.
6. Перечень спецификаций запросов, подлежащих ручному программированию.

Заголовок сервера. Заголовок сервера должен соответствовать имени указанному в головном файле и в названии самого файла:

server <имя сервера> (<список конфигурационных переменных>);

В заголовке сервера в скобках может быть задан список конфигурационных параметров, загружаемых из конфигурационного файла сервера. Эти параметры доступны в серверных программах.

Перечень описаний таблиц базы данных сервера. В этом разделе описывается схема базы данных, с которой будет работать данный сервер. Имя базы данных генератором присваивается автоматически - *<имя сервера>_dbs*. Также автоматически присваиваются и другие необходимые реквизиты БД. Основным

содержательными объектами реляционной базы данных являются отношения или таблицы. Таблица определяется конечным набором полей (столбцов). Описание таблиц базы данных в серверном файле имеет следующий вид:

table <имя таблицы>
(<тип поля><имя пол[,<тип поля><имя поля]…
):<Первичный ключ таблицы>;< опции таблицы>

Типы полей таблиц задаются в терминах проектных типов данных. Для таблицы может быть определен первичный ключ, используемый для организации связей между таблицами. Опции таблицы определяют полное ее наименование, а также список простейших запросов к таблице (select, insert, update, delete, cursor), для которых должны быть автоматически сгенерированы соответствующие программные компоненты.

Перечень индексов базы данных. Здесь к введенным таблицам определяются все необходимые индексы. Индексы могут быть уникальными и не уникальными. Если в таблице определен первичный ключ, то соответствующий набор полей должен быть указан среди уникальных индексов таблицы.

index <имя индекса> on <имя таблицы> [/unique]
(<список полей индекса>);

Перечень (набор) связей между таблицами базы данных. В этом разделе задаются связи между таблицами базы данных. Данная информация эквивалентна заданию пар primary/foreign key. Она используется при автоматической генерации запросов для поддержания целостности базы данных.

set <имя набора> owner <имя таблицы- владельца набора>
memeber <имя таблицы - членов набора>
(<список полей ключевого индекса>);

Перечень описаний SQL-запросов, используемых в программах сервера. SQL-запросы к базе данных, связанные с основными операциями над одной таблицей (select, insert, update, delete и cursor), могут быть сгенерированы системой без описания

простым указанием специальных опций при описании таблицы (смотри выше). В данном разделе описываются все остальные SQL-запросы, которые исполняются в программах сервера. Запросы снабжаются именами и совокупностью входных и выходных формальных параметров. Описание запроса содержит заголовок с идентификатором запроса и списком формальных параметров, за которым следует собственно тело запроса. В теле запроса используется стандартный язык запросов к реляционным базам данных SQL. Для каждого описанного запроса генерируется совокупность С-функций, вызов которых из программ сервера предусматривает исполнение соответствующих запросов к базе данных.

sql <имя запроса>
([<список входных параметров>]):
([<список выходных параметров>])
< запрос на языке SQL>;
<опции к запросу>

Подмножество языка SQL, которое используется в данном контексте, достаточно представительно для того, чтобы эффективно манипулировать данными.

Перечень спецификаций запросов к прикладному серверу, подлежащих ручному программированию. В данном разделе перечисляются спецификации запросов к серверу, программы которых разработчик предусматривает написать сам с использованием языка С. Спецификация содержит идентификатор (имя) запроса, совокупность входных параметров и совокупность выходных параметров, а также многочисленные опции.

link <имя ручного запроса>
([<список входных параметров>]):
([<список выходных параметров>])
/source = <имя С-файла >;
<другие опции к запросу>

Опция */source*, которая задаёт имя С-файла, является обязательной. Для каждого ручного запроса автоматически

генерируется в Н-файл для прототипов функций самого запроса, а также функций получения входных параметров и отсылки выходных параметров. Предусматривается также генерация файлов-заготовок, в которых размещаются образцы запросов. В программах ручных запросов могут вызываться функции исполнения SQL-запросов. В составе опций запроса отметим признак однозначного результата, т.е. декларация, что запрос выдает не таблицу, а простую совокупность параметров. Эта опция влияет на вид интерфейса клиентского модуля при использовании данного запроса. Запрос может быть снабжен перечнем привилегий пользователя, которые будут проверяться перед его исполнением. Задается также перечень совокупностей запросов, в которые он входит для использования в других проектах.

Описание клиентских модулей. Каждому клиентскому модулю, упомянутому в файле описания проекта, соответствует отдельный mdl-файл описания модуля. Этот файл содержит заголовок модуля (*module <имя модуля>;)* и список спецификаций процедур. Процедура в описании модуля представляет основное интерфейсное понятие. Спецификация процедур состоит из нескольких разнотипных компонент и в зависимости от их состава в каждой конкретной процедуре, ей соответствует тот или иной вид экрана приложения - модуля.

Описание каждой процедуры клиентского модуля состоит из перечисленных ниже разделов.

1. Заголовок процедуры.
2. Вызов функции динамической библиотеки -1.
3. Описание диалога ввода данных.
4. Вызов функции динамической библиотеки -2.
5. Описание формата метки процедуры в дереве.
6. Список вызовов запросов к серверам.
7. Список вызовов процедур и запросов.

Обязательным разделом является только заголовок.

Заголовок процедуры. Здесь задаются идентификатор процедуры, список ее формальных параметров, наименование и ряд опций.

proc <имя процедуры>
(<список формальных параметров>):
<строка комментария>;
<опции к процедуре>

Процедуры в описании модуля могут вызывать друг друга по имени. Порядок описания процедур в модуле не зависит от порядка их вызова. Допускается прямая или косвенная рекурсия при вызове процедур. В сгенерированной программе в процессе работы создается дерево вызовов процедур. Каждому узлу этого дерева соответствует свой экземпляр процедуры. При создании очередного экземпляра процедуры ему передаются фактические параметры, которые запоминаются как атрибуты соответствующего узла дерева вызовов.

Вызов функции динамической библиотеки - 1. Создание экземпляра процедуры может предусматривать вызов некоторой функции из динамической библиотеки. Спецификации таких функций задаются в файле описания проекта и содержат имя функции, входные и выходные формальные параметры. Вызов таких функций в теле процедуры может осуществляться в двух местах – сразу после заголовка и после ввода данных. Вызов функций имеет следующий вид:

func <имя функции>
(<список входных фактических параметров>):
(<список выходных фактических параметров>);

В случае вызова – 1 (сразу после заголовка) в качестве соответствующих входных фактических параметров могут быть заданы либо константы проекта, либо формальные параметры процедуры, значения которых при создании экземпляра процедуры на момент вызова функции уже известны. Выходные фактические параметры должны быть уникальны в пределах процедуры. Их

значения вместе с формальными параметрами процедуры выдаются в первом окне процедуры.

Описание диалога ввода данных. Создание экземпляра процедуры может сопровождаться вводом данных пользователем. Реализация ввода в генераторе осуществляется через диалоги. Данный раздел содержит спецификации составляющих элементов диалога:

get
(<элемент ввода> <опции к элемент ввода>
[,<элемент ввода> <опции к элемент ввода>]…
):<строка комментраия>;

Список элементов ввода имеет иерархическую структуру, позволяющую задать некоторую дисциплину ввода информации. На верхнем уровне иерархии может быть задана совокупность закладок (диалог с закладками или Property Sheet). Далее идет список элементов ввода/вывода. Вид элементов диалога зависит от описания соответствующих переменных и их типов. Это может быть простое окно ввода текста с возможной текстовой пометкой справа от него. Если задан перечислимый тип, то элемент может быть представлен в виде combobox или radio buttons. В качестве очередного элемента может быть задана совокупность элементов, взятая в скобки с заданием опций всей такой группы. Такой конструкции соответствует стандартный элемент groupbox. Каждый элемент при необходимости может быть снабжен рядом параметров и признаков, влияющих в ограниченных пределах на размещение в окне диалога. Элементы могут быть снабжены ссылками на доступные в данном контексте другие переменные описания, которые будут задавать начальные значения.

Вызов функции динамической библиотеки - 2. Данный вызов отличается от аналогичного вызова перед спецификацией ввода тем, что он выполняется сразу после завершения ввода данных пользователем. В качестве входных фактических параметров функции могут использоваться переменные, заданные при вводе.

Описание формата метки процедуры в дереве. Каждому созданному экземпляру процедуры соответствует узел дерева вызовов процедур модуля. Кроме фактических параметров для обеспечения наглядности каждому узлу дерева вызова могут быть в качестве атрибутов узла присвоены пиктограммы и тестовые комментарии. Пиктограмма задается в опции в заголовке процедуры. А текстовый комментарий задается в данном разделе и представляется в следующем виде:

label <форматная строка> <параметры комментария>;

В составе параметров комментария могут использоваться параметры процедуры, выходные параметры функций и вводимые параметры.

Список вызовов запросов к серверам. В процедуре можно задать произвольное количество запросов к прикладным серверам. Каждый такой запрос состоит из имени запроса, списка входных фактических параметров и списка выходных параметров:

link <имя запроса к серверу>
([<список входных фактических параметров>]):
([<список выходных фактических параметров>]);
< опции к запросу>

Здесь могут использоваться любые специфицированные в проекте запросы: автоматические запросы, которые специфицируются соответствующими опциями при таблицах, специфицированные sql – запросы и специфицированные ручные запросы. Для клиентских модулей не важно, в каких серверах были специфицированы эти запросы, поскольку с помощью одного клиентского модуля можно связываться с разными серверами. Запросы выполняются при создании экземпляра процедуры после вызова динамических функций и завершения ввода. Результаты вызовов функций и параметры, вводимые в разделе get, могут использоваться в качестве входных фактических параметров вызовов серверных запросов. Запросы группируются по серверам. Каждая группа запросов, адресованная одному серверу, в заданном порядке пересылается на сервер для исполнения в контексте одной

транзакции. Сервер отсылает ответ для каждого запроса в виде либо таблицы, либо в виде набора параметров (в зависимости от спецификации запроса).

Список вызовов других процедур и запросов. Как уже говорилось, каждая процедура может допускать вызовы к другим процедурам. Здесь различаются два типа вызовов – вызов произвольных процедур данного модуля и вызов специальных, «молчаливых» серверных запроса, то есть таких, которые не имеют выходных параметров.

call [<описание диалога>]
{<имя процедуры>|link <имя запроса>}
(<список фактических параметров>):
<строка коментария>;
<опции к вызову>

Вызов может сопровождаться вводом данных (как при вызове процедур, так и при вызове серверных запросов). Для этого запрос снабжается описанием диалога, аналогичным описанию ввода в начале процедуры. Фактическими параметрами вызовов могут быть любые переменные, упомянутые в предыдущих разделах описания процедуры.

WEB – сайты. Каждому WEB-сайту, упомянутому в головном файле проекта, соответствует отдельный файл описания этого сайта (расширение .sit). Это описание содержит заголовок сайта и спецификации основных его объектов, которыми являются страницы (*page*) и действия (*action*):

Описание страниц. Понятие страницы связано с представлением на WEB-сайте информации полученной в результате выполнения запросов к прикладному серверу через ИНТЕРНЕТ. Описание отдельной страницы содержит заголовок с формальными входными параметрами и список вызовов серверных запросов для формирования таблиц, которые являются результатом выполнения запросов. Страницы являются функциональным аналогом процедур в клиентских модулях, но имеют иные синтаксис и семантику.

page <имя страницы >(<формальные параметры>):

<строка комментария>;
<опции к странице>
[form [image <имя файла рисунка>] =
{page<имястраницы>|action<имядействия>}(<парметры>);]...
link <имя запроса>
(<фактические входные параметры запроса>):
(<выходные параметры запроса>);
<опции к запросу>
[form [image <имя файла рисунка>] =
{page<имястраницы>|action<имядействия>}(<парметры>); }]...

Разделы *form* определяют гиперссылки на другие описанные в этом сайте страницы или действия. Заголовок и таблицы могут быть снабжены гиперссылками на другие описанные страницы и действия. В гиперссылках могут задаваться входные фактические параметры. Описание страниц может сопровождаться многочисленными опциями, влияющими на размещение текста в HTML-странице, задание различных рисунков, в том числе управляемое выходными данными запроса. Тем не менее, весь этот механизм макетирования действует только в том случае, когда WEB-мастер не предусмотрел собственного шаблона для данной страницы. В этом случае и генерируется шаблон в соответствии с заданными здесь опциями, который, затем, может быть модифицирован Web-мастером.

Описание действий. Действия - это реакция на активизацию соответствующих гиперссылок, размещенных на страницах сайта. Действия, как правило связаны с выполнением специфицированных в проекте запросов и дальнейшим вызовом необходимых страниц для представления результатов запросов.

action <имя действия>(<формальные параметры>):
<строка комментария>;
<опции к действию>
{link <имя запроса>
(<фактические входные параметры запроса>):

(<выходные параметры запроса>);
<опции к запросу>}...
show <имя страницы> (<передаваемые параметры >);
error <имя страницы> (<передаваемые параметры >);

Пользователь, работающий с WEB-сайтом с помощью стандартного ИНТЕРНЕТ-броузера, активизируя необходимые гиперссылки, вызывает выполнение описанных запросов. Гиперссылки реализуются на WEB-сервере с помощью автоматически генерируемых программы-демона и программ CGI-интерфейса. Именно программа-демон при запуске осуществляет присоединение к прикладному серверу (серверам) и сохраняет контекст безопасности. Для прикладного сервера программа-демон является клиентским модулем и работает с точки зрения безопасности от имени некоторого зарегистрированного пользователя с определенными полномочиями. Автоматически сгенерированные CGI-программы запускаются заново для каждого запроса и не содержат никакого контекста безопасности. При каждом запуске CGI-программа осуществляет присоединение к демону, работающем на том же компьютере, для передачи запроса и получении ответа. При нормальном выполнении запроса информация передается на страницу, указанную в разделе show, а при выдаче ошибки в процессе исполнения запроса вызывается страница, указанная в разделе error.

Использование в прикладной системе WEB-интерфейсов носит публичный характер. Поэтому здесь применяется своя система безопасности, отличная от применяемой подсистемы при связи модулей (демонов) с прикладными серверами.

Генерация и сборка системы

Программа генератора проектов реализована в виде консольного приложения для платформ Win32 и Linux. Для генерации проекта необходимо запустить программу в директории проекта и указать в качестве параметра идентификатор проекта, а так же набор опций для указания требуемых платформ. Имеется также оконный вариант в виде диалога, в котором можно задать опции в визуальной форме и запустить генерацию нажатием на кнопку диалога (смотри рисунок 4).

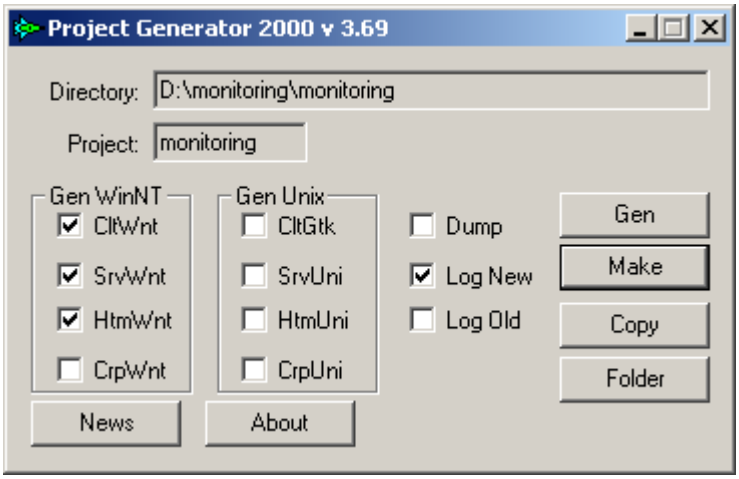

Рис. 4. Окно «Генератора проекта»

Предусмотрена генерация следующих платформ.

- Cltwnt – клиентские модули для Win32,
- Cltgtk – клиентские модули для Linux (gtk-библиотека),
- Srvwnt – прикладные серверы для Win32,
- Srvuni – прикладные серверы для Linux,
- Htmwnt – WEB-сайт для Win32 (MS IIS 3/4),
- Htmuni – WEB-сайт для Linux (Apache),

- Crpwnt – программные файлы подсистемы безопасности для Win32,
- Crpuni - программные файлы подсистемы безопасности для Linux.

Вместе с текстами программного кода генерируются скрипты для сборки программ проекта по исходным текстам в соответствии с выбранной платформой. Кроме программ предусмотрена генерация разнообразной справочной информации о проекте, например, экспертная подсистема сообщает о наличии в директориях проекта файлов с неизвестным назначением, о различных излишествах в описании проекта (неиспользованные параметры, запросы и пр.).

Для подсистемы безопасности предусмотрена генерация отдельного модуля администратора безопасности, с использованием того же самого механизма, что и для обычного модуля. Для этого модуля в составе каждого сервера предусмотрен набор запросов ведения списка пользователей, разделения их на группы с заданием состава полномочий каждой группы. Предописанные системные запросы подсистемы безопасности могут, при желании, использоваться и в других модулях.

Механизм экспорта серверных запросов предусматривает генерацию специальных библиотек, которые можно встраивать в клиентские программы, написанные без использования генератора проектов. Эти библиотеки обеспечивают взаимодействие с сервером в соответствии с требованиями подсистемы безопасности.

Результатом сборки системы является полный комплект исполняемых файлов системы для каждой из используемых в проекте платформ. Например, для платформы Srvwnt будут в общем случае собраны следующие программные компоненты:

<имя сервера>.exe – исполняемые файлы прикладных серверов системы. Это основные компоненты системы. Они выполняют все запросы, поступающие с клиентских модулей.

< имя сервера >_sqlcnf.exe, < имя сервера >_sqlconf.exe – конфигураторы базы данных. Конфигураторы предназначены для

подготовки скриптов по инсталляции новых баз данных, по модификации структуры данных при доработках системы, по разгрузке и загрузки содержимого баз данных системы.

crpsetup.exe – конфигуратор подсистемы безопасности. Конфигуратор системы безопасности предназначен для загрузки или модификации информации в базу данных системы безопасности.

Кроме перечисленных исполняемых модулей в системе собирается также ряд динамически загружаемых библиотек.

- crplib.dll – библиотека алгоритмов шифрования,
- nslcon.dll – консольная библиотека сетевого уровня системы безопасности,
- nslwin.dll – оконная библиотека сетевого уровня системы безопасности,
- srvgate.dll – серверная интерфейсная библиотека системы безопасности,
- zlib.dll – библиотека компрессирования информации,
- db_dblb6.dll – драйвер интерфейса с СУБД MS SQL Server 6.5,
- db_dblb7.dll – драйвер интерфейса с СУБД MS SQL Server 7.0,
- db_odbc.dll – драйвер интерфейса с произвольной СУБД через ODBC,
- db_orcl.dll – драйвер интерфейса с СУБД Oracle 7,
- db_sybase.dll – драйвер интерфейса с СУБД Sybase 11.

Некоторые из перечисленных библиотек собираются только в том случае, если указаны соответствующие опции. Кроме перечисленных программных компонент в исполняемый комплект платформы входят конфигурационные файлы серверов:

- <имя сервера>.conf,
- <имя сервера>_crp.conf.

Для платформы Cltwnt собираются все исполняемые клиентские модули и ряд библиотек:

- <имя клиентского модуля>.exe – исполняемые файлы клиентских модулей,
- sysadm.exe – исполняемый файл модуля администратора,
- <имя проекта>_utl.dll – библиотека утилит (экспорт данных, передача файлов и пр.),
- crplib.dll – библиотека алгоритмов шифрования,
- nslcon.dll – консольная библиотека сетевого уровня подсистемы безопасности,
- nslwin.dll – оконная библиотека сетевого уровня подсистемы безопасности,
- cltgate.dll – клиентская интерфейсная библиотека подсистемы безопасности,
- zlib.dll – библиотека компрессирования информации

В состав комплекта исполняемых программ входят конфигурационные файлы:

- <имя клиентского модуля >.conf.

Пример описания проекта информационной системы

Приведем пример описания микро проекта прикладной информационной системы. Данный пример носит чисто методический характер и не является прототипом реально действующей системы. Тем не менее, на этом примере можно проследить основные идеи структурной модели информационных систем, положенные в основу описываемого здесь проектного подхода. Рассмотрим небольшую информационную систему, которую условно назовем “Системой мониторинга курса валюты”. Под задачей мониторинга здесь мы понимаем сбор, хранение и анализ ряда наблюдаемых величин, характеризующих некоторую совокупность объектов. В качестве наблюдаемых объектов в данном примере рассматриваются курсы валют. Анализ и прогноз изменения курса иностранной валюты является вожделенной задачей для многих финансовых структур и не только для структур, но и для многих наших соотечественников. Курсы валют обычно используются в разных целях, поэтому «Система мониторинга» в

части хранения курсов валюты должна быть открыта для других систем.

Общая структура «Системы мониторинга курса валют» показана на рисунке 5.

Рис. 5. Структура системы

Описание проекта этой системы представлено в файлах:

monitoring.gen,
monitoring.typ,
currency.srv,
operator.mdl.

Головной файл проекта *monitoring.gen* содержит следующий текст:

```
project monitoring :"МОНИТОРИНГ КУРСА ВАЛЮТ";
  /version=2
  /baseport=47888
  /database=(wnt_dblb7)
username (t_usr,t_usr_name,t_usr_dn),(t_grp,t_grp_name,t_grp_stat);
icon  monitoring,operator,currency,del,list,new,upd,tab;
server currency;
module operator:"МОДУЛЬ ОПЕРАТОРА ВВОДА";
```

Основная информация о проекте заключена в последних двух строчках этого файла и состоит в том, что система состоит из одного сервера *currency* и одного клиентского модуля *operator.* Остальная же часть головного файла содержит детали. В опциях к заголовку проекта указан текущий номер версии проекта, номер TCP/IP порта для сервера, а также указано, что в данном проекте

будет использован драйвер для работы через библиотеку ntwdblib с MS SQL Server /7.0.

Следующий раздел, начинающийся с ключевого слова *username*, связан с формированием подсистемы безопасности. Здесь определяется структура реквизитов пользователей системы и реквизиты групп пользователей. Среди этих реквизитов выделим статус пользователей группы (t_grp_stat), который задает режимы доступа к информации. Если приведенные в разделе *username* типы не описаны в проекте, то они создаются автоматически и используются при создании базы данных пользователей.

В разделе *icon* перечислены имена пиктограмм, которые используются в проекте. Заметим, что обязательно должны быть указаны одноименные пиктограммы, для всех специфицированных в проекте серверов и модулей. Каждой пиктограммы, упомянутой в этом разделе, должен соответствовать файл *<имя пиктограммы>.ico* в директории *icons* проекта.

Файл *monitoring.typ* содержит описание всех типов переменных которые используются в проекте:

```
type t_cur : char(char[4])
  /title="Код валюты"/hdr="Код валюты"
type t_iso : char(char[10])
  /title="Код валюты ISO"/hdr="ISO"
type t_cur_name : char(char[100])
  /title="Наименование валюты"/hdr="Наименование"
type t_cur_simb : char(char[4])
  /title="Обозначение валюты"/hdr="Обознчение"
type t_count : numb(short)
  /title="за"/hdr="Количество единиц базовой валюты"
type t_rate : money(double)
  /title="Курс валюты"/hdr="Курс"
type t_date : date(int)
  /title="Дата"/hdr="Дата"
```

Следующий файл *currency.srv* содержит полное описание прикладного сервера *currency:*

```
server currency();

table r_cur
  ( t_cur  a_cur,
    t_iso   a_iso,
    t_cur_name  a_cur_name,
    t_cur_simb   a_cur_simb
  ):a_cur;
 /title="Список применяемых валют"
 /link_insert /link_update /link_cursor /link_delete


table r_tab_cur
  ( t_cur a_cur,
    t_date a_date,
    t_count  a_count,
    t_rate   a_rate
 ):a_cur,a_date;
  /title="Курсовые таблицы валюты"
  /link_insert /link_delete

index i_cur_1 on r_cur/unique(a_cur);
index i_tab_cur_1 on r_tab_cur/unique(a_cur,a_date);

set s_tab_cur_1 owner r_cur member r_tab_cur(a_cur);

sql tab_cur(t_cur cur):(t_date date,t_rate rate,t_count count)
   cursor for select
      a_date,a_rate,a_count
   from
      r_tab_cur
   where
      a_cur = :cur;
  /link
```

/title="Таблица курса выбранной валюты"

База данных сервера состоит из двух таблиц: r_cur – списка зарегистрированных валют, и r_tab_cur – собственно таблицы курсов валют, которые ведутся по датам для каждой из зарегистрированных видов валюты. При каждой таблице указан первичный ключ и ряд опций. В частности опции */link_cursor, /link_insert, /link_update, /link_delete* указывают на необходимость автоматического создания программных функций для реализации основных запросов к базе данных (cursor, insert, update, delete). После описания таблиц идет список используемых индексов, в том числе и упомянутых выше первичных ключей. Раздел, начинающийся со слова set, устанавливает связь между таблицами r_cur и r_tab_cur по ключевому индексу. Наконец, последняя конструкция, начинающаяся со слова sql, представляет собой описание SQL – запроса, для которого необходимо сгенерировать соответствующие программные функции.

Приведенное описание сервера содержит всю необходимую информацию для автоматического создания прикладного сервера. Для обеспечения пользовательского интерфейса служит клиентский модуль *operator.* Ниже приведен текст соответствующего файла:

```
module operator;

proc tab_cur(t_cur cur):"ТАБЛИЦА КУРСОВ ВАЛЮТЫ";
/icon = list
   link tab_cur_cursor(cur):(a_date,a_rate,a_count);
   call insert_tab_cur(cur):"Новая строка";
     /icon=new
     /update
   call link r_tab_cur_delete(cur,a_date):"Удаление строки";
     /confirm="Подтвердите удаление"
     /icon=del
```

```
proc insert_tab_cur(t_cur cur)
:"ВВОД ТЕКУЩЕГО КУРСА ВАЛЮТЫ";
/icon=new
/break
    get (t_date date,t_rate rate/newline,t_count count):"Ввод данных";
   link r_tab_cur_insert(cur,date,count,rate):();

proc list_cur():"СПИСОК ВАЛЮТ";
   link r_cur_cursor():(a_cur,a_iso,a_cur_name,a_cur_simb);
   call "Ввод"
  ( t_cur cur,t_iso iso,t_cur_name cur_name,t_cur_simb cur_simb)
    link r_cur_insert(cur,iso,cur_name,cur_simb):"Новый вид валюты";
      /update
      /icon=new

  call "Ввод"
  (*a_cur,t_iso iso/init=a_iso,t_cur_name cur_name/init=a_cur_name,
    t_cur_simb cur_simb/init=a_cur_simb)
  link r_cur_update(a_cur,iso,cur_name,cur_simb):"Модификация";
    /update
    /icon=upd

 call link r_cur_delete(a_cur):"Удаление";
    /confirm="Подтвердите удаление"
   /update
   /icon=del

 call tab_cur(a_cur):"Таблица курсов выбранной валюты";
```

Модуль operator состоит из спецификаций трех процедур. Каждая из этих процедур обращается к определенному запросу и может опционально вызывать другие процедуры или запросы к прикладному серверу. Фактически, в тексте модуля представлено дерево возможных вызовов:

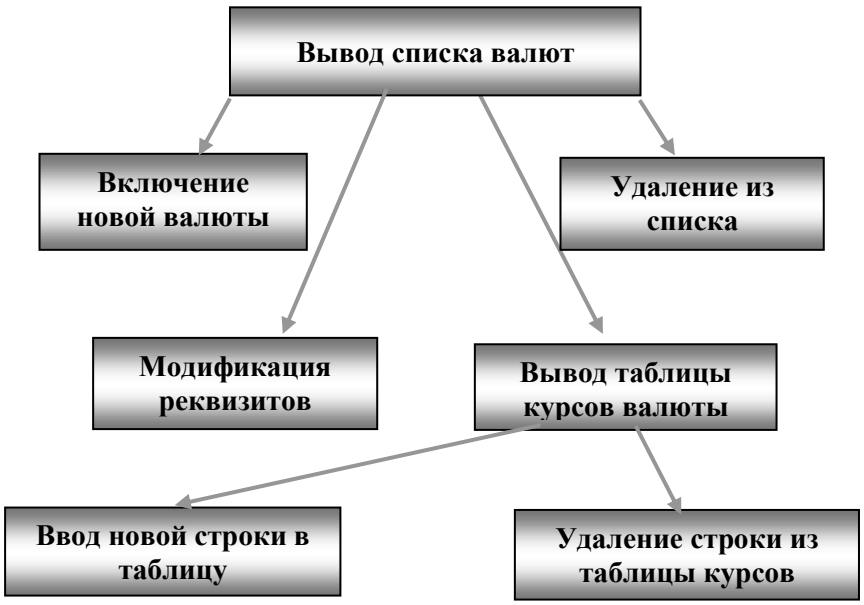

Рис. 6. Структура дерева возможных вызовов в модуле operator

Различные опции при процедурах и вызовах позволяют реализовать удобный интерфейс и дисциплину работы с модулем.

Если говорить о реализации данного проекта в целом, то после генерации программного кода и после сборки, мы получим следующий состав исполняемых файлов:
currency.exe – прикладной программный сервер,
currency_sqlconf.exe – оконный конфигуратор базы данных,
currency_sqlcnf.exe – консольный конфигуратор базы данных,
crpsetup.exe – конфигуратор системы безопасности,

operator.exe – клиентский модуль,
sysadmin.exe – модуль администратора системы,
а также конфигурационные файлы и необходимые динамические библиотеки.

После конфигурации системы безопасности и первичной загрузки БД пользователей, а также после создания и конфигурирования прикладной базы данных может быть запущен прикладной сервер currency.exe. После запуска сервера необходимо с помощью модуля администратора системы ввести реальный список пользователей и определить их полномочия. После этого система готова к работе, то есть к запуску пользователями единственного клиентского модуля системы operator.exe. С одним прикладным сервером может одновременно работать несколько операторов.

Клиентские модули системы работают в рамках обычного оконного интерфейса. Работающий модуль представлен окном со стандартными компонентами – menu, toolbar, основное окно, statusbar (см. рисунок 7).

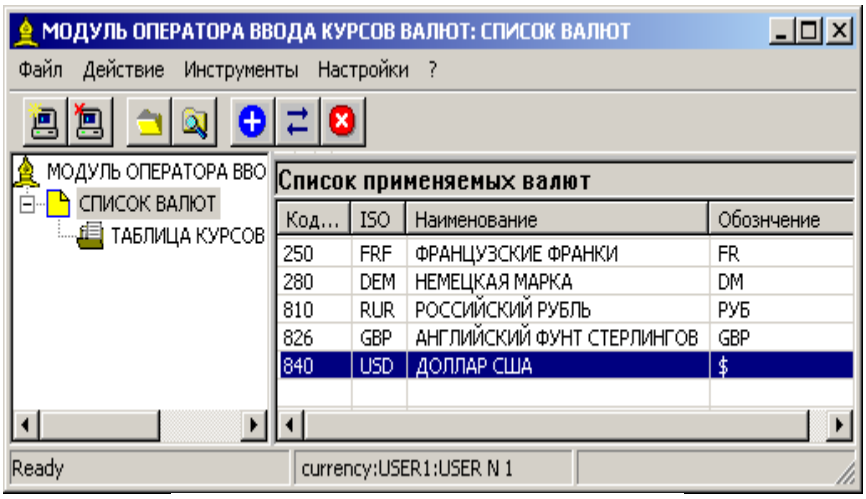

Рис. 7. Окно модуля оператора ввода

Основное окно содержит две панели: слева дерево вызовов задействованных на данный момент процедур, справа совокупность таблиц, соответствующих той процедуре дерева вызовов, которая в данный момент активизирована. Состав меню, toolbar и statusbar зависит от текущей процедуры. Пользователь имеет возможность в любой момент открыть дополнительные окна указанного вида. Эти окна работают в контексте одного процесса и используют общий контекст безопасности. В каждом окне будет выстроено собственное дерево обращений к процедурам. Кроме того, пользователь может открывать окна без левой древовидной панели, оставляя больше места для таблиц.

Меню клиентского модуля содержит пункты, позволяющие присоединяться к прикладному серверу и отсоединяться от него. Процесс присоединения модуля к серверу (открытие сеанса связи) предусматривает взаимодействие с подсистемой безопасности. Пользователь должен предъявить требуемые пароли и сертификаты, для того, чтобы доказать, что он имеет право доступа к серверу. Процесс соединения с сервером обеспечивает уверенность пользователя, что он соединился именно с требуемым сервером, а не с его подделкой.

Средствами клиентского модуля можно экспортировать информацию текущей процедуры (параметры, таблицы) в формат HTML, MS Word, MS Excel (пункт меню Инструменты). Для таких преобразований используются специальные шаблоны, которые опытный пользователь может настраивать для получения требуемых изобразительных возможностей. При отсутствии шаблона для данного типа процедуры, он создается автоматически, и в последствии может быть скорректирован пользователем.

Пользователю предоставляется возможность изменить состав и порядок параметров, а также состав и порядок колонок каждой таблицы в правой части окна (пункт меню Настройки). Можно также изменить пропорции размеров таблиц в пределах правой панели. Все эти настройки сохраняются в конфигурационном файле модуля и используются при повторном вызове приложения.

Щелчки мыши по заголовкам колонок изменяют вид сортировки таблицы.

Можно было бы и дальше обсуждать детали спроектированной системы, однако очевидно, что подробностей реальной системы слишком много, чтобы излагать их в относительно небольшой публикации. В то же время содержательное описание проекта вместилось в четыре маленьких текстовых файла общим объемом около ста строк.

Практическое использование Генератора проекта

Эффективность всякой автоматизации, как правило, измеряется коэффициентом повышения производительности труда. Разумеется, и при применении нашего проектного подхода к разработке прикладных информационных систем хотелось бы иметь такие оценки. Мы могли бы конечно привести цифры, тем более, что это не трудно сделать, сравнив объем программного кода и объем проекта системы. В частности, для рассмотренного выше примера такое соотношение составит величину второго порядка. То есть, объем автоматически сгенерированного программного кода (сумма объемов только *.c, *.h файлов ~0.7 Мгб) более чем в сто раз превышает суммарный объем исходных файлов проекта (~5 Кбт). Для крупных проектов это соотношение будет поменьше, но все равно оно достигает величины, как минимум, порядка десятка. Ниже дано краткое описание тех прикладных проектов, которые были разработаны в разные годы средствами Генератора проектов.

1992:
Система обработки почтовых и телеграфных авизо (Заказчик ГУ ЦБ РФ по г. Москве). Генеральный подрядчик: АО «Телеформ».

Автоматизированная Система обработки почтовых и телеграфных авизо, была внедрена и эксплуатировалась в течение нескольких лет в Главном Расчетно-кассовом Центре ГУ ЦБ РФ по г.Москве и Шаболовском РКЦ.

1993:
Система приема и обработки электронных межбанковских платежей "АС МБР" (Заказчик: ГУ ЦБ РФ по г. Москве). Генеральный подрядчик: АО «Телеформ».

Система обработки межбанковских платежей в режиме реального времени была внедрена и эксплуатировалась в течение нескольких лет в Главном Расчетно-кассовом Центре ГУ ЦБ РФ по г.Москве и Шаболовском РКЦ..

1993:
Проект платежной системы на основе смарт-карт для Сбербанка России. Генеральный подрядчик: BGS Industrials.

Был разработан пилотный проект платежной системы на основе смарт-карт для Сбербанка России, прообраз системы Автоматизированной Системы Расчетов по Пластиковым Картам СБ РФ (АС РПК).

1994:
Пилотный проект интегрированной системы автоматизации банковской деятельности для Владимирского банка СБ РФ.Генеральный подрядчик: Digital Equipment Corp., Россия

Были выполнены постановка задачи и разработка пилотного проекта системы автоматизации межбанковских и межфилиальных расчетов для Владимирского банка Сбербанка России.

1996 – 1999:
Система автоматизации банковской деятельности коммерческих банков и Учреждений Сбербанка РФ. (Заказчик: Башкирский Сбербанк РФ). Генеральный подрядчик: компания Банковский Производственный Центр, позднее – компания «АйТи».

Система комплексной автоматизации многофилиального универсального коммерческого банка (ГАМБИТ) разработана, внедрена и эксплуатируется в Башкирском банке Сбербанка России.

Система автоматизирует деятельность Главной бухгалтерии банка, ОПЕРУ, Территориального расчетного центра, отделений банка. Подробное описание системы дано в [8].

1998 - 1999
Система урегулирования взаимной задолженности субъектов экономической деятельности РПБ КЛИРИНГ (Заказчик: КБ «Роспромбанк»).

Автоматизированная система представляет собой программно-технологический комплекс, предназначенный для проведения работ по взаимному многостороннему урегулированию просроченной задолженности субъектов экономической деятельности на региональном и федеральном уровнях.

1998 - 1999
Система «ЭЛЕКТРОННЫЙ КАТАЛОГ ДЕТАЛЕЙ ОБЪЕКТОВ МАШИНОСТРОЕНИЯ» (Заказчик: ЗАО «Русавиа»).

Автоматизированная система «ЭЛЕКТРОННЫЙ КАТАЛОГ ДЕТАЛЕЙ ОБЪЕКТОВ МАШИНОСТРОЕНИЯ» представляет собой программно-технологический комплекс, предназначенный для хранения и оперативного доступа к данным, которые описывают каталоги узлов, агрегатов и деталей разнообразных сложных технических объектов машиностроения.

1999:
Система аудита биллинга сотовых операторов. (Заказчик: Группа страховых компаний НАСТА).

Данная система позволяет провести внешний аудит счетов, выставляемых операторами сотовой связи своим клиентам. Система позволяет загрузить базы данных клиентов, тарифных планов, трафика и счетов клиентов из системы биллинга сотового оператора для их анализа и проверки.

1999:
Пилот-проект «Electronic Transcribing System». (Заказчик: «BrainStorm Engineering, LLC» , США). Генеральный подрядчик: ЗАО «СмартКарт Сервис».

Прототип многопользовательской автоматизированной системы для распределенной обработки медицинской информации через Internet.

1999 – 2000:
Проект eCommerce. (Заказчик: Сбербанк РФ). Генеральный подрядчик: ЗАО «СмартКарт Сервис».

Был разработан проект и сертифицирован проект электронной коммерции. Разработана Система Интернет-Платежей, которая позволяет производить оплату товаров и услуг в Интернет-магазинах в режиме реального времени с помощью микропроцессорных карт СБЕРКАРТ Сбербанка России.

2000 – 2001:
Макет проекта Manager. (Заказчик: ЗАО «СмартКарт Сервис»). Генеральный подрядчик: ЗАО «СмартКарт Сервис».

Был разработан макет системы «Менеджер», которая предназначена для управления экономической и финансовой деятельностью предприятия. В разработке системы «Менеджер» наиболее существенным является то, что здесь управление копредприятием рассматривается с точки зрения управления проектами, которые это предприятие ведет.

2002-2003:
Проект системы Duplet. (Заказчик: **ООО «Про-Кард»*).***
Генеральный подрядчик: ЗАО «СмартКарт Сервис».

Разработан и внедрен проект автоматизированной системы DUPLET, предназначеной для обеспечения Интернет торговли и проведения некоторых банковских операций в режиме реального времени с микропроцессорными картами стандарта DUET (карts

СБЕРКАРТ Сбербанка России), эмитированных платежными системами с разными генеральными ключами.

2002-2003:
Проект системы мобильного банкинга. (Разработчик системы: ЗАО «СмартКарт Сервис»).

Система мобильного банкинга предназначена для предоставления клиентам коммерческих банков услуг безопасного управления банковским счётом с использованием личного мобильного телефона. Для обмена данными между банком и клиентом используется служба коротких сообщений (SMS).

***

Это неполный перечень тех прикладных информационных систем, которые были реализованы с помощью Генератора проектов. Безусловно Генератор обеспечивает очень высокую производительность для разработчиков прикладных систем. Однако даже не в конкретных цифрах увеличения производительности труда разработчиков состоит основной эффект от примения Генератора проекта. Главное преимущество проектного подхода состоит в том, что он не перемешивает две области деятельности – содержательное моделирование прикладных задач и технологические проблемы программирования. Благодаря этому появляется возможность достаточно глубокой формализации на этапе постановки задачи, то есть в процессе проектирования системы. С другой стороны средства проектирования, то есть правила описания системы, их форма и содержание, однозначно соответствуют доступной в данный момент технологии создания программ, которая заложена в Генераторе проекта и той структурной модели, на которую опирается Генератор. Развитие структурной модели, расширение возможностей автоматической генерации программ будут совершенствовать и средства проектирования. В связи с этим нам представляется, что дальнейшее углубление проектного подхода, разработка более мощных языковых средств описания

содержательных объектов, моделей и процедур, может быть даже их специализация и ориентация на конкретные предметные области, являются весьма актуальной задачей для разработчиков прикладных информационных систем.

Литература

1. David A. Marca, Clement L. McGowan. SADT: Structured Analysis and Design Technique. McGrow-Hill Book Company, New York, 1988.
2. Brackett, J., and C. McGowan: “Applying SADT to Large System Problems”, SofTech Technical Paper TP059, January 1977.
3. SofTech, Inc. “Introduction to IDEF0”, SofTech Deliverable no. 7500-14, September 1979.
4. Вышинский Л.Л., Прибытков Ю.Д., Шиленко В.И., Широков Н.И. Инструментальная система ФАКИР. Известия Академии наук СССР, Техническая кибернетика, Москва, 1986 г., № 3.
5. Вышинский Л.Л., Гринев И.Л., Шиленко В.И., Широков Н.И. Инструментальные средства САПР. В сб. Задачи и методы автоматизированного проектирования в авиастроении. Издание Вычислительного Центра АН СССР, Москва, 1991 г.
6. Гринев И.Л., Широков Н.И. Средства управления данными в САПР. В сб. Задачи и методы автоматизированного проектирования в авиастроении. Издание Вычислительного Центра АН СССР, Москва, 1991 г.
7. Вышинский Л.Л., Гринев И.Л., Демидов А.Ю., Широков Н.И. Технологии разработки и сопровождения АБС. «Банковские технологи», Москва, 1997 июль-август.
8. Вышинский Л.Л., Гринев И.Л., Катунин В.П., Лабутин И.В., Флеров Ю.А. Широков Н.И. Банковские Информационные технологии (части I и II). Издание Вычислительного Центра РАН, Москва, 1991 г.