РОССИЙСКАЯ АКАДЕМИЯ НАУК
ВЫЧИСЛИТЕЛЬНЫЙ ЦЕНТР
им. А.А. Дородницына РАН

АВТОМАТИЗИРОВАННОЕ ПРОЕКТИРОВАНИЕ ФИНАНСОВЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

ВЦ РАН
Москва, 2004

УДК 519.86

Ответственные редакторы
доктор физ.-матем. наук *Ю.А. Флёров*,
кандидат физ.-матем. наук *Л.Л. Вышинский*

В сборнике статей представлены работы, посвящённые автоматизации проектирования сложных информационно-поисковых и информационно-расчётных финансовых систем.

Статьи этого сборника содержательно разделяются на две части. В первой части обсуждаются теоретические проблемы, возникающие при разработке САПР финансовых приложений. Описывается проектный подход к созданию таких систем и инструментальные программные средства для реализации такого подхода.

Во второй части даны краткие описания практических финансовых информационных проектов и реальных систем, которые реализованы с использованием разработанных в Вычислительном центре им. А.А. Дородницына РАН инструментальных средств.

Работа выполнена при частичной финансовой поддержке Минобрнауки (контракт № 37.011.11.0019) и программы фундаментальных исследований Президиума РАН «Математическое моделирование и интеллектуальные системы» (проект 2.31).

Рецензенты: *И.Г. Поспелов, А.М. Попов*

Научное издание

Автоматизация проектирования информационных систем

П.С. Краснощёков, Л.Л. Вышинский, Ю.А. Флёров

Одной из основных задач автоматизации проектирования как предметной области является разработка инструментальных программных средств, обеспечивающих повышение эффективности разработки проектов, и технологии создания сложных объектов различного целевого назначения. Безусловно, специфика проектируемых объектов определяет перечень конкретных решаемых в системе автоматизированного проектирования (САПР) задач, а также её архитектуру, математическое, программное, информационное и техническое обеспечение. Традиционный класс объектов автоматизации проектирования обычно связывают со сложными техническими объектами в машиностроении, электронике, строительстве и других отраслях промышленности. В рамках этого класса объектов достаточно хорошо проработаны теоретические основы САПР [1,2], а также существует целая гамма конкретных реализаций систем автоматизированного проектирования. Вычислительный центр РАН принимал активное участие в создании САПР летательных аппаратов.

В 1991 году в издательстве ВЦ РАН был издан сборник статей «Задачи и методы автоматизированного проектирования в авиастроении». С тех пор круг наших интересов в области автоматизации проектирования существенно расширился. Нами был получен ряд теоретических и практических результатов в области автоматизации проектирования информационных систем для различных финансовых приложений [3-6]. Казалось бы, такое изменение предметной области должно привести к полному изменению средств и методов автоматизации проектирования. Но оказалось, что многие проблемы синтеза сложных систем (а информационно-поисковые и информационно-расчётные финансовые системы очень сложны) инвариантны по отношению к предметной области. Поэтому мы решили опубликовать ряд полученных нами результатов в этой области в новом сборнике с похожим названием «Задачи и методы автомати-

зированного проектирования финансовых информационных систем» и тем самым подчеркнуть общность задач и преемственность результатов.

Проектирование и разработка современных информационных систем в области финансовых приложений является очень дорогостоящим делом. Характерным примером сложных финансовых приложений являются автоматизированные банковские системы (АБС). Разработка АБС, их внедрение в работу банков, последующая доработка, доводка, сопровождение могут занимать не один год работы больших коллективов высококлассных специалистов. Закономерности современного этапа развития сферы финансовых услуг характеризуются очень высокими темпами роста объёма услуг, расширением спектра финансовых инструментов, усложнением применяемых технологий. Это влечёт за собой усложнение разрабатываемых финансовых информационных систем. Программные банковские системы конца 80-х гг., так называемые «операционные дни» банков, как правило, были автономными программами и обеспечивали лишь подготовку и печать ежедневных отчётных форм. Современные клиент - серверные системы с сотнями и тысячами удалённых рабочих мест обрабатывают в режиме реального времени миллионы транзакций в сутки. Практически возникла новая отрасль, связанная с разработкой финансовых информационных приложений.

Динамика расширения области финансовых приложений и жёсткая конкуренция в этой сфере диктуют необходимость кардинального сокращения сроков их разработки. Для достижения этой цели разрабатывающие компании вынуждены привлекать большое число программистов. Однако рост числа разработчиков, увы, не приводит к сокращению сроков разработки. И если ещё сроки доведения системы до сдачи в эксплуатацию можно как-то регулировать директивным способом, то доводка до рабочего состояния сложных финансовых информационных систем растягивается на неопределённое время и приводит к существенным расходам разрабатывающих компаний.

Эта ситуация очень напоминает период бума научно-технической революции в области создания систем вооружения, когда сложность различных классов технических объектов бурно рос-

ла. При этом и сроки проектирования, и особенно сроки натурных испытаний, объём доработок, модификаций катастрофически увеличивались, несмотря на большой приток научных и технических кадров в эти области. Может быть, наша аналогия покажется натянутой, но если сравнить, например, самолёт с несколькими десятками тысяч деталей и современную банковскую программную систему, состоящую из сотен тысяч операторов, то станет очевидной, по крайней мере, сопоставимость проблемы роста энтропии при проектировании самолётов и при разработке сложных информационных систем. Энтропию проектов сложных разрабатываемых систем можно оценить как

$$H = -\sum_{i=1}^{N} \left(p_i \log \; p_i + (1 - p_i) \log \; (1 - p_i) \right),$$

где N – это характеристика сложности проекта системы (число деталей в проекте самолёта или число операторов в исходном программном коде АБС), а p_i – вероятность того, что в i-м элементе проекта (в чертеже детали или в тексте оператора) разработчик допустил ошибку. В применении к проектированию энтропия – это величина противоположная качеству, это мера неадекватности созданного изделия проекту. Действительно, энтропия пропорциональна числу вопросов, которые необходимо «задать» созданной по этому проекту системе, чтобы определить её состояние, т.е. выявить и устранить допущенные в процессе проектирования ошибки. Увеличение энтропии всегда очень дорого стоит. При проектировании самолётов это ведёт к увеличению объёма лётных испытаний. В применении к финансовым информационным системам увеличение энтропии означает увеличение затрат и времени на доведение системы до рабочего состояния, а зачастую и необходимость существенных доработок в процессе их функционирования. Доработки, модификации, обновление версий финансовых систем в процессе эксплуатации являются очень неприятным, трудоёмким и дорогим последствием увеличения энтропии исходных проектов. Фирмам - разработчикам АБС и других аналогичных систем приходится держать специальные службы сопровождения, трудозатраты которых, как правило, в несколько раз превышают трудозатраты разработчиков проекта.

Из сказанного выше очевидно, что увеличение численности разработчиков проекта без изменения подхода к проектированию не может существенным образом повлиять на сроки и качество создания систем, а зачастую и ухудшает его, поскольку увеличивается вероятность ошибок при усложнении координации разработки проекта. Единственным выходом из этой ситуации является уменьшение сложности проекта и уменьшение вероятности ошибок проектирования. Только таким образом можно существенно уменьшить значение энтропии проекта и, следовательно, не только уменьшить трудоёмкость разработки, но и существенно сократить затраты сопровождения системы.

Уменьшение сложности проекта – это не уменьшение сложности разрабатываемой системы. Увы, сложность системы – это неизбежность развития современных финансовых технологий. Уменьшение сложности проекта – это упрощение формального описания создаваемого программного продукта, т.е. уменьшение того самого N – количества «операторов» исходного текста проекта системы. Мы заключили слово «оператор» в кавычки потому, что уж если говорить о формальном языке описания проекта, то это должен быть непроцедурный язык высокого уровня, в котором основными элементами являются не операторы, а описания математических моделей, содержательных понятий и объектов, связанных с предметной областью. Проект – это формальный документ или совокупность формальных документов, которые обладают определённой структурой, определённым составом своих компонент, правилами оформления, синтаксисом, семантикой и прочими атрибутами формального объекта. Существенное требование к языку описания проектов состоит в том, чтобы уровень описания и формализации проекта был бы достаточным для однозначного истолкования его на следующем этапе, на этапе создания программного кода.

Другим резервом уменьшения энтропии проекта является уменьшение вероятности возникновения ошибок. Самым эффективным способом реализации этой задачи в программных системах является автоматизация создания программного кода. Создание программного кода фактически является технологическим этапом реализации проекта и если проект – это формальное описание разраба-

тываемой системы, то создание программного кода может быть и должно быть полностью автоматизировано. Автоматизация создания программного кода требует специального инструментария – Генератора программного кода, который является технологической компонентой проекта. Жизнь проекта программной системы не заканчивается её реализацией. Длительность жизненного цикла программных систем определяется способностью к модификациям. Как уже говорилось выше, модификация работающей системы является весьма дорогим и сложным процессом. Если при разработке системы используется автоматический Генератор программного кода, то процедура модификации существенно упрощается за счёт того, что изменения вносятся только в проект, а все дальнейшие изменения в программном коде, информационном окружении и прочих компонентах системы осуществляются автоматически.

В настоящем сборнике представлены работы, которые направлены на реализацию сформулированных выше целей и задач. Статьи этого сборника разделены на две содержательные части. В первой части обсуждаются теоретические проблемы, возникающие при разработке САПР в данной предметной области. А во второй – даны краткие описания практических финансовых информационных систем, которые реализованы, с использованием разработанных в Вычислительном центре РАН инструментальных средств. Авторы сборника приносят глубокую благодарность ЗАО «СмартКард-Сервис» и лично его Генеральному директору А.В. Камышникову за всестороннюю поддержку практической реализации представленных здесь работ.

ПРОЕКТНЫЙ ПОДХОД К РАЗРАБОТКЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Л.Л. Вышинский, И.Л. Гринёв, Ю.А. Флёров, Н.И. Широков

Понятие проекта прикладной информационной системы требует своего разъяснения. В данном контексте в это понятие вкладывается смысл технического, инженерного описания разрабатываемой системы, а не бизнес-проекта процесса её создания. Это весьма существенно, поскольку вопросы, связанные с организацией «проекта», с финансированием «проекта», с управлением «проектом», и другие подобные вопросы разработки информационных систем здесь рассматриваться не будут. Тем не менее для уяснения смысла, который мы связываем с понятием проекта системы, нам потребуется рассмотреть этапы жизненного цикла прикладных информационных систем.

Жизненный цикл информационных систем охватывает все стадии её создания, сопровождения и развития:

- анализ и разработка требований к системе;
- проектирование системы, в том числе
  - разработка архитектуры системы,
  - разработка математических моделей предметной области,
  - разработка и анализ макета системы (эскизный проект),
  - разработка технического проекта;
- разработка (программирование) системы;
- интеграция, сборка и испытания системы;
- эксплуатация системы, сопровождение, модернизация.

Современные действующие стандарты не предлагают конкретных методов реализации всех этих этапов. Существуют различные подходы к организации работ, которые связаны с разными моделями жизненного цикла. Наиболее распространёнными являются каскадная и спиральная модели разработки прикладных информационных

систем. На рис. 1 и 2 схематически показано различие между этими подходами.

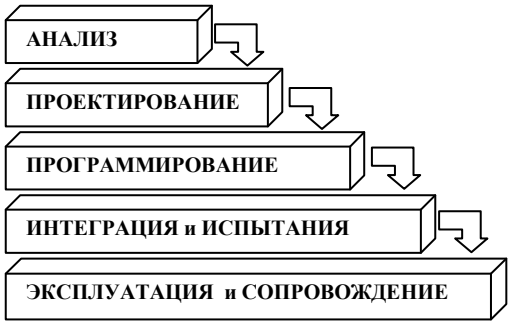


Рис.1 Каскадная модель

Каскадная модель жизненного цикла информационных систем отражает схему разработки в виде последовательности завершённых этапов. Данная модель характерна для достаточно полно специфицированных систем, эксплуатация которых предполагается в неизменных условиях, а доработки сводятся к исправлениям ошибок программирования и незначительным корректировкам проектных решений.

Рис.2 Спиральная модель

Спиральная модель жизненного цикла характерна для развивающихся систем. Эта модель предусматривает возможность их модернизации. При этом модернизация системы затрагивает все стадии разработки: анализ новых возникших задач, проектирование и про-

граммирование новой версии системы. Такая модель в большей степени приспособлена к решению задач современных финансовых приложений с их достаточно высокой динамикой развития. Преимущество спиральной модели для разработки информационных систем очевидны, но они могут быть в полной мере реализованы только при высокой степени автоматизации этапов проектирования, разработки и сопровождения системы. Сейчас на рынке программных продуктов существует большое количество инструментальных программных средств автоматизации разработки программных приложений, так называемых CASE – средств (Computer-Aided Software Engineering). Обычно к CASE-продуктам относят любое программное средство, автоматизирующее те или иные этапы жизненного цикла. В разряд CASE-средств попадают как дешёвые программы с весьма ограниченными возможностями, позволяющие рисовать блок-схемы простеньких программ, так и очень дорогие программные комплексы, при реальном применении которых в создании конкретных приложений используется от силы несколько процентов от их возможностей. К сожалению, приходится констатировать, что не существует универсальных программных продуктов, которые могли бы быть эффективно использованы для автоматизации разработки на всех этапах жизненного цикла любых информационных приложений.

В связи с этим, возникает такое ощущение, что наиболее рациональным способом автоматизации разработки сложных прикладных информационных систем, является создание для каждого класса систем или даже для отдельных приложений проблемно-ориентированного инструментария, который бы в наибольшей степени соответствовал решаемым задачам. Если следовать уже проводимым ранее аналогиям с проектированием самолётов, то для их производства специально проектируется и создаётся оснастка – стапели, шаблоны, сборочный инструмент и прочее. Ровно так же для больших информационных систем нужно иметь специальные инструментальные средства автоматизации, которые были бы неотъемлемой технологической компонентой проекта системы. Понятно, что такой проблемно-ориентированный инструментарий не может и не должен обладать многими свойствами универсальных CASE-

продуктов, но он должен решать те задачи автоматизации, без которых невозможно обойтись при разработке конкретных приложений.

Какими же свойствами должны обладать подобные «встраиваемые» в проект инструментальные средства? Ниже перечислены основные задачи автоматизации, которые приходится решать при проектировании информационных систем:

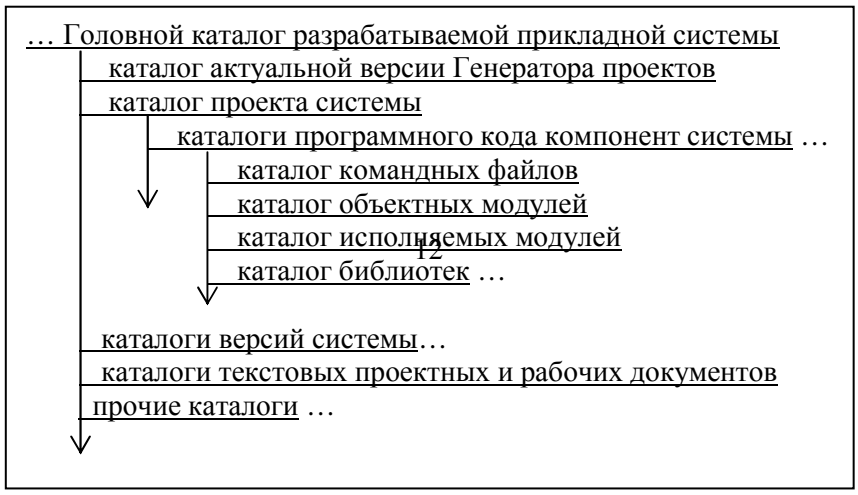
- создание единого информационного пространства разрабатываемого проекта;
- описание архитектуры проектируемой системы;
- структурное проектирование информационных систем;
- разработка математических моделей объектов предметной области;
- описание логических структур и моделей хранимых данных;
- описание серверных компонент системы;
- описание пользовательского интерфейса, механизмы редактирования пользовательских документов, форм, диалогов и прочее;
- создание опытных образцов, макетов, демонстрационных стендов;
- автоматизация программирования, генерация исходного программного кода;
- автоматизация разработки программной и эксплуатационной документации системы;
- автоматизация разработки программы, методики и тестов функциональных, ресурсных и других испытаний системы;
- автоматизация разработки средств загрузки и инсталляции системы, средств, обеспечивающих системное обслуживание и эксплуатацию.

Все эти задачи в той или иной степени приходится решать при разработке финансовых информационных систем. Ниже дано более подробное описание перечисленных задач. При этом мы здесь будем ориентироваться в основном на свой опыт решения этих задач, который был приобретён при разработке многих финансовых (и не только финансовых) приложений. Как уже говорилось, с каждым

проектом мы связываем некоторый специальный инструментарий, который затем становится неотъемлемой частью проекта. Нами был разработан ряд программных продуктов с общим названием Генератор проектов. Каждый новый финансовый проект оставлял свой отпечаток на этом инструментарии. Возникали новые задачи, новые проблемы автоматизации – в ответ на это возникали новые возможности инструментария, новые версии Генератора проектов.

Единое информационное пространство проекта (не путать с Единым информационным пространством объекта автоматизации!) необходимо для хранения созданной в процессе разработки системы информации и обеспечения коллективного доступа к этой информации. В рамках единого информационного пространства хранятся все документы от технического задания на разработку и вплоть до исполняемых модулей и инсталляционных пакетов всех разработанных версий. Средствами инструментария необходимо поддерживать целостность хранимой информации, т.е. осуществлять контроль полноты и непротиворечивости данных, соответствие дат, версий и других атрибутов различных компонент разрабатываемого проекта. Создание единого информационного пространства проекта – это первый формальный шаг, с которого начинается разработка прикладной информационной системы.

В рамках Генератора проектов единое информационное пространство содержит полное описание проекта, полный состав исходного программного кода по всем компонентам системы, все версии системы, текущую рабочую, проектную, программную и другую документацию в электронном виде, а также любую другую информацию, которая необходима при разработке и эксплуатации системы. В наших проектах структурная организация единого информационного пространства, как правило, имеет следующую структуру каталогов:



Как было сказано выше, Генератор проектов, вернее, его текущая версия, является неотъемлемой частью разрабатываемого проекта. Основная информация о системе содержится в каталоге проекта системы. По мере разработки проект пополняется проектными документами, которые представляют собой файлы определённого типа. Каждая из перечисленных выше задач проектирования – это либо новый проектный файл, либо новая сущность, которая добавляется в уже созданные проектные документы.

Открытие нового проекта начинается с присвоения проекту имени или идентификатора. Это важный этап, поскольку идентификатор проекта будет в дальнейшем многократно использоваться в процессе разработки, эксплуатации, сертификации и пр, пр. Например, имена каталогов единого информационного пространства могут быть определённым образом связаны с идентификатором проекта. Кроме идентификатора обычно при открытии нового проекта задаётся ряд обязательных и необязательных атрибутов – дата открытия проекта, организация – разработчик, организация – заказчик, ответственные исполнители, ссылки на техническое задание, другие документы, а также любая другая информация, полезная при проектировании, выпуске проектной документации и разработке системы.

Архитектура проектируемых систем – это один из главных вопросов, который нужно решить на самых первых шагах проектирования информационных систем. Под архитектурой понимается состав основных компонент информационной системы, и связи между ними. Существенной особенностью финансовых информационных систем является то, что они, как правило, являются многопользовательскими клиент-серверными системами, работающими с большими объёмами хранимой информации. Основными информационными и программными компонентами таких систем, представленными на рис .3, являются:

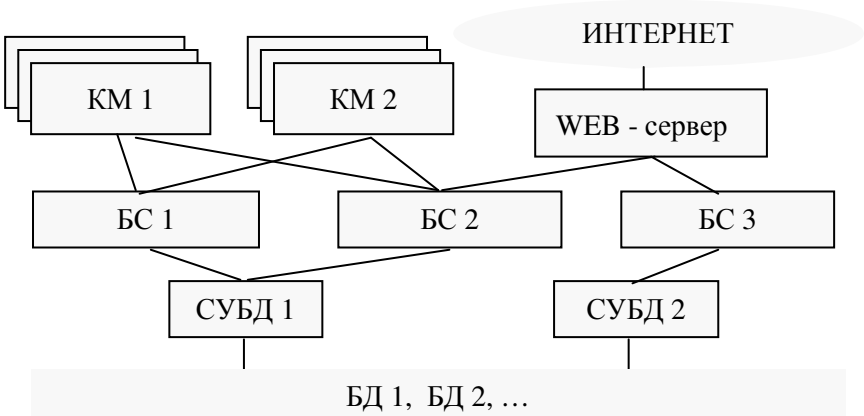


Рис. 3. Клиент – серверная архитектура

- БД – централизованные или распределённые базы данных, другие хранилища информации;
- СУБД – системы управления базами данных, другие программные средства, которые организуют физический доступ к данным, организуют и обеспечивают надёжное их хранение, поиск, модификацию;
- БС – прикладные бизнес-серверы, программы, которые принимают запросы от пользователей системы на выполнение определённых бизнес-процедур, обрабатывают эти запросы, проводят необходимые вычисления, формируют необходимые логические запросы к базам данных, получают информацию из баз данных, формируют и отправляют ответы пользователям;
- КМ – клиентские модули, программы, которые, обеспечивают формирование запросов пользователей на выполнение необходимых бизнес - процедур, передачу этих запросов на БС, приём ответов от БС и представление полученных результатов на терминальных устройствах;
- WEB-серверы – прикладные программы, которые обеспечивают санкционированный доступ к прикладной системе пользователям ИНТЕРНЕТ. WEB-серверы являются специализированными клиентскими модулями системы;
- программные и информационные средства, обеспечивающие связь между отдельными компонентами системы.

Для задания архитектуры проекта информационной системы нужно задать конкретный состав программных компонент и возможные связи между ними. При этом задаются идентификаторы, типы, наименования и другие атрибуты компонент системы.

Структурное проектирование. Для задания архитектуры системы, а также при решении других задач могут быть применены методы и средства структурного проектирования. Структурное проектирование представляет собой создание системы связанных между собой различного типа информационных диаграмм. В области структурного проектирования выработаны определённые стандарты и виды представления различных объектов, отношений и процессов (см. [7-9]):

- SCD (диаграммы структурных схем),
- ERD (диаграммы типа «сущность-связь»),
- DFD (диаграммы потоков данных),
- DSD (диаграммы структур данных),
- SAD (диаграммы, описывающие архитектуру),
- CSD, DTD, MSD, BSD, …

Существует ряд готовых инструментальных систем, которые автоматизируют структурное проектирование прикладных информационных систем и оперируют геометрическими изображениями разных типов диаграмм, стрелок указателей, бирок и прочее. Целью структурного моделирования является построение спецификаций основных функций разрабатываемой системы, её архитектуры, состава компонент, структуры данных, блок-схем и других объектов в наглядной и доступной форме. Некоторые инструментальные системы структурного моделирования позволяют по структурной модели создавать работающие макеты для проверки основных концепций принимаемых проектных решений. Однако главной целью этого этапа является стандартизация на уровне проекта принятых технических решений. Это позволяет поддерживать высокий уровень исполнения проектной дисциплины и в результате обеспечивает повышение качества и надёжности разрабатываемых информационных систем.

Впрочем, сразу нужно сказать, что альтернативой структурным моделям могут быть хорошо проработанные на начальных этапах проектирования традиционные функциональные спецификации различных объектов. В Генераторе проектов нет встроенных программных средств структурного проектирования.

Математическое моделирование объектов предметной области. Это наиболее сложная и наиболее ответственная задача всего процесса проектирования прикладных систем. Не существует и не может существовать универсальных методов и средств математического моделирования произвольных систем, объектов и процессов. В то же время для конкретных классов задач существуют специализированные средства и языки моделирования предметной области. Именно это обстоятельство и позволяет предполагать, что наиболее эффективными системами автоматизации проектирования могут

быть проблемно-ориентированные CASE – средства. При разработке финансовых приложений необходимо уметь моделировать такие сущности как:

- субъекты финансовых отношений и их организационные структуры;
- финансовые и платёжные инструменты;
- финансовые операции и технологии их выполнения;
- механизмы и технологии бухгалтерского учёта;
- аналитические инструменты финансовой деятельности.

Для реализации математических моделей этих понятий достаточно простых языковых средств, которые описывают различные структуры данных подобных объектов, конечные связи между атрибутами этих объектов и некоторые отношения, определяемые на их множестве. Множество значений произвольного конечномерного объекта может быть представлено в следующем виде:

$$X = \{ x_1, x_2, \cdots, x_n;\ где x_i \in X_i;\ P(x_1, x_2, \cdots, x_n)\}.$$

Здесь X_i - описанные ранее объекты, а P – предикат, определяющий множество допустимых значений $x \in X$. Отношение между объектами задаётся аналогично:

$$R(y_1, \cdots, y_m;\ y_i \in Y_i) = \{r_1, \cdots, r_k; r_i \in R_i; P_R(y_1, \cdots, y_m, r_1, \cdots, r_k)\},$$

где P_R – предикат, задающий отношения R на декартовом произведении множеств $Y_1, \cdots, Y_m$, а r_i – свойства этого отношения.

Данные конструкции фактически задают основу для непроцедурного языка высокого уровня, на котором можно описывать модели приведённых выше объектов и понятий для финансовых систем.

Существует два подхода к реализации математических моделей, т.е. к использованию их в бизнес-процедурах. Первый подход связан с построением на базе этих моделей программ, реализующих конкретные задачи предметной области. А другой подход связан с механизмом интерпретации математических моделей в конкретных условиях решаемых задач. Выбор способа реализации математиче-

ских моделей представляет собой один из аспектов разработки архитектуры системы.

Логические структуры данных формируются на основе разработанных для проектируемой системы математических моделей. Логические структуры используются при организации передачи и хранения данных. Как правило, хранимая информация в системе отражает текущее состояние и параметры моделей объектов, с которыми она имеет дело, а также любые архивные, справочные, нормативные, настроечные и другие данные, необходимые для её функционирования. Характер использования хранимой информации определяет *модель хранимых данных.* Наибольшее распространение в банковских технологиях получили реляционная, иерархическая и сетевые модели данных. Большинство промышленных баз данных в настоящее время ориентируются на реляционную модель данных, которая обладает рядом очевидных преимуществ – это простота реализации, простота пользовательского интерфейса, наличие удобного и простого языка запросов. Однако для задач, которые требуют организации сложных структур данных и разнообразных неоднородных процедур обработки, реляционная модель данных может быть недостаточно эффективна. В этом случае целесообразно выбрать сетевую модель. Иерархическая модель является частным случаем сетевой. Генератор проектов допускает работу и с реляционными и с сетевыми структурами данных.

В проекте системы должны быть явно указаны имена и типы баз данных, в которых предполагается хранить информацию, а также должны быть описаны логические структуры этих баз данных.

Описание логических структур данных реляционной модели представляет собой набор реляционных таблиц, полей, индексов и других объектов баз данных. Для описания сетевых данных кроме описания таблиц (в сетевой терминологии – записей) необходимо задание отношений между записями типа «один ко многим» – наборов. Кроме таблиц и наборов в описание логических структур включаются спецификации различных запросов для поиска, создания и модификации хранимой информации. В этих спецификациях указываются имя запроса, параметры и характер выполняемых действий. Таким образом, с каждой базой данных системы связывается множе-

ство функций манипуляции хранимыми данными. Эти функции реализуются средствами СУБД и обеспечивают интерфейс прикладных бизнес-серверов с базами данных.

Описание прикладных бизнес-серверов. В прикладной информационной системе может быть один или несколько бизнес - серверов. Прикладные серверы обеспечивают связь пользователей с базами данных и выполнение пользовательских запросов. Каждый сервер должен быть поименован, и должны быть указаны его возможные связи с базами данных. Для каждого прикладного сервера описывается полный состав выполняемых им бизнес-процедур. Реализация бизнес-процедур основывается на этих описаниях, на описании запросов к базам банных и на описании математических моделей предметной области. Основанием для выполнения бизнес-процедур является пользовательский запрос, который представляет собой входной документ определённой структуры или несколько связанных между собой документов. Результат выполнения пользовательского запроса также представляет собой один или несколько выходных документов. На рис. 4. показана общая схема выполнения любого пользовательского запроса.

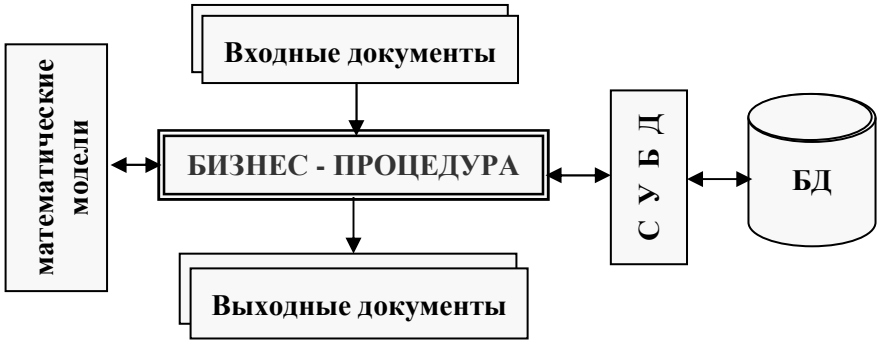


Рис. 4. Схема выполнения бизнес-процедур прикладным сервером

Входные и выходные документы представляют определённым образом организованную, структурированную информацию. Структура документов определяется их типом. Типы используемых документов должны быть описаны на этапе построения логических

структур данных. Бизнес-процедуры в процессе исполнения обращаются к математическим моделям, которые могут быть реализованы в виде библиотек программ или использоваться в режиме интерпретации.

Описание пользовательского интерфейса. Исполнение бизнес-процедур инициируется пользователями, которые посылают на серверные компоненты свои запросы с помощью клиентских модулей. Подробные описания клиентских модулей, декларированных в архитектуре системы, должны быть представлены в проекте системы. Основными объектами описания клиентских модулей являются пользовательские запросы и сценарии (последовательность) их вызовов. Структура пользовательского запроса содержит ссылки на серверные бизнес-процедуры, ссылки на типы входных и выходных документов бизнес-процедур, способы формирования входных документов и формы интерпретации выходных. Других функций у клиентских модулей нет. Таким образом, описание клиентского модуля представляет собой описание пользовательского интерфейса. Пользовательский интерфейс определяется:

- составом пользовательских запросов к прикладным серверам,
- структурой пользовательских меню и команд на всех уровнях клиентских модулей,
- представлением входных и выходных документов в виде экранных форм, диалогов разного типа и твёрдых копий,
- сценариями формирования и анализа экранных документов,
- структурой диалогов для формирования документов,
- представлением входных и выходных документов в виде форматированных файлов.

Для представления всех этих компонент проекта должны быть специальные средства, т.е. должен быть реализован специальный язык описания пользовательского интерфейса.

На этом можно закончить представление основных разделов проекта прикладной информационной системы, по крайней мере, в рамках того проектного подхода, который связан с технологией Генератора проектов, и подвести промежуточный итог. Полноценный

проект прикладной информационной системы должен содержать описание

- архитектуры системы,
- совокупности используемых математических моделей,
- логической структуры и моделей данных,
- прикладных серверов и реализуемых ими бизнес-процессов,
- клиентских модулей и реализуемых ими пользовательских запросов к прикладным серверам.

Никакой мало-мальски сложный проект не создаётся в одночасье. Как правило, рабочему проекту информационной системы предшествует её эскизный проект. На основании эскизного проекта может быть создан макет, прототип системы. Под макетом прикладной информационной системы понимается действующий комплекс программ заданной в проекте архитектуры, в котором реализованы все или основные программные компоненты системы, выполняющие некоторую часть своих функций. Может быть, эти функции представлены в урезанном, ограниченном виде, но в любом случае макет – это реальная система, в определённой степени адекватная текущей стадии эскизного проекта. Только после апробации на макете основных принятых технических решений на уровне архитектуры, математических моделей, структур данных, бизнес-процедур и пользовательского интерфейса можно судить об их адекватности поставленной задаче.

Разработка программного кода макета (как и разработка рабочей версии системы) представляет собой технологический этап создания прикладной системы. В рамках концепции Генератора проектов эти технологические этапы должны быть полностью автоматизированы. Благодаря полноте описания системы на этапе проектирования Генератор проектов имеет возможность автоматически генерировать полный исходный программный код для всех компонент системы и для всех необходимых программных или информационных утилит, которые обеспечивают интеграцию, сборку, инсталляцию, конфигурирование и эксплуатацию системы. Концепция автоматической генерации программного кода подразумевает использование Генератором проектов большого количества макросов, раз-

личных заготовок программного текста для типовых программных конструкций и решений.

На рис. 5. показана схема реализации проектного подхода, положенного в основу Генератора проектов. Эта схема полностью соответствует спиральной модели разработки прикладных систем, поскольку для внесения изменений в программный код необходимо вернуться в самое начало разработки – к проекту системы. Создание новых версий при этом существенно упрощается, так как не требует ручного исправления программного кода.

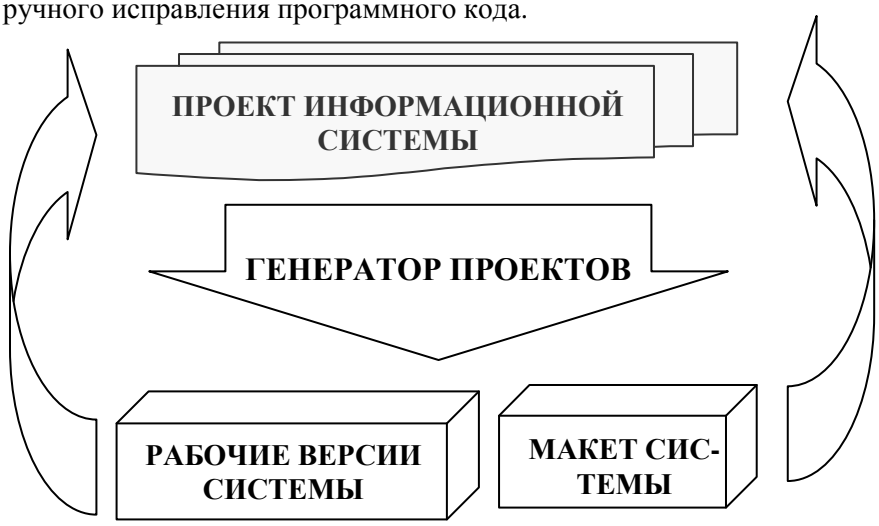


Рис. 5. Схема генерации макета и рабочей версии системы

Эффективное применение такой схемы предполагает существенную разницу в сложности, т.е. в объёме описания, проекта системы и сложности рабочего программного кода. Имеющийся опыт применения Генератора проектов при разработке многих финансовых приложений показывает, что объём описания проекта на порядок, а то и на два, меньше, чем объём программного кода всех компонент системы.

ГЕНЕРАТОР ПРОЕКТОВ

Н.И. Широков

Генератор проектов предназначен для автоматизации разработки программных комплексов, использующих следующие информационные технологии:

- многоуровневая клиент - серверная архитектура;
- удалённые пользовательские рабочие места;
- режим реального времени выполнения бизнес-процедур;
- использование реляционных и сетевых структур данных;
- оконный интерфейс пользовательских приложений;
- доступ к бизнес-процедурам через ИНТЕРНЕТ - сайты;
- защита информации от несанкционированного доступа.

Основным механизмом автоматизации является автоматическая генерация текстов программ по формальному описанию проекта разрабатываемой системы.

Мотивация разработки генератора

Генератор проектов был разработан и впервые применён в реальных разработках в начале девяностых годов. Тогда пришлось реализовывать несколько больших систем в банковской области, в которых была использована технология клиент - сервер. В процессе работы над этими системами возникли проблемы поддержания целостности программной реализации при неизбежно возникающих изменениях в структурах данных и алгоритмах их обработки. Необходимо было проводить согласованные правки в большом количестве программных компонент. По мере реализации подобных проектов всё более и более становилось понятно, что большинство программ как на сервере, так и на клиенте, написаны по типовому шаблону и отличаются друг от друга количеством и составом полей и наличием или отсутствием некоторых типовых ситуаций. Это означало, что разрабатываемые программы укладываются в некоторую формальную модель. Поэтому возникла мысль описать модель системы на формальном языке и по её описанию автоматически генерировать тексты программ. Такой подход обеспечивал очевидное сокращение

объёма написанного разработчиком кода, так как размер описания модели на порядки меньше соответствующего сгенерированного кода. Второе не столь очевидное, но очень важное преимущество использования модели состояло в том, что по мере усложнения системы и добавления в неё новых функциональных возможностей ранее написанные программы можно автоматически модифицировать в соответствии с новыми использованными в модели штампами.

Эта идея была реализована в виде инструментальной системы, которая получила название Генератор проектов. В дальнейшем в течение вот уже более чем десяти лет подавляющее большинство проектов мы разрабатывали с использованием Генератора, который развивался и усложнялся от проекта к проекту. В результате сформировался так называемый «проектный подход» к разработке информационных систем. Этот подход позволил с минимальными усилиями перенести множество проектов с MS DOS под MS Windows 3.11, с протокола DECNET на протокол TCP/IP, с VAX/VMS на Windows NT 3.51, с Windows NT 3.51 на ALPHA/AXP под OSF (DecUnix), , с СУБД RDB/VMS на MS SQL Server, на Oracle и т.д. и т.п. Менять приходилось, в основном, только некоторые программы Генератора. В дальнейшем были освоены новые клиентские платформы, использующие графический пользовательский интерфейс

Если в общем оценивать роль Генератора в разработке информационных систем, то здесь уместна следующая аналогия. Разработка программ на традиционных языках с использованием известных библиотек напоминает строительство зданий. Обилие инструментальных средств – различные оконные интерфейсы (MFC, Delphi, Gtk), и прочие полезные библиотеки – напоминает магазин стройматериалов с огромным ассортиментом. Обилие различных материалов и приспособлений наряду с их доступностью вовсе не гарантирует успешность и быстроту постройки здания. Для этого необходимо, во-первых, навыки проектировщика и строителя, а, во-вторых, время. Использование Генератора проектов в этой аналогии можно сравнить с домостроительным комбинатом, производящим типовые панели, блоки и прочие типовые компоненты, из которых по типовым проектам различных серий можно строить дома с большой скоростью. Вряд ли можно таким образом возвести храм Василия Бла-

женного или Кёльнский собор, но добротный микрорайон с развитой инфраструктурой вполне реально построить и даже очень быстро.

Этапы разработки программ

При разработке любого программного комплекса достаточной сложности прослеживаются четыре основных фазы его реализации: 1) постановка задачи, 2) формализация математических моделей и методов решения частных задач, 3) программирование и, наконец, 4) сборка и получение дистрибутивов.

Постановка задачи, техническое задание являются результатом совместной работы заказчика и разработчика.

Вторая фаза, которая является собственно созданием формального проекта разрабатываемого комплекса, это результат работы аналитиков. Аналитик изучает поставленную задачу, формализует её и предлагает методы решения. Разработка проекта на основе постановки задачи (1→2) – процесс в значительной степени творческий. Здесь чрезвычайно важен опыт аналитика и его общематематическая подготовка, а также знакомство с современными информационными технологиями. Этот процесс по своей сути неформальный и вряд ли будет автоматизирован в обозримое время.

Получение текстов программ из описания проекта (2→3) требует использования квалифицированного персонала – программистов. В значительной степени это связано с тем, что описание проекта бывает недостаточно формализованным. Последнее обстоятельство, в свою очередь, обусловлено огромной пропастью между весьма примитивным уровнем современных языков программирования и потребностью в математических формализмах при постановке задачи. На этапе формализации задачи аналитик может манипулировать абстрактными понятиями – совокупность, таблица, отношение. В свою очередь программист должен выбрать для каждого конкретного понятия в проекте тот или иной метод представления (реализации). Для каждого выбранного метода приходится использовать или реализовывать подходящие программные средства.

Получение дистрибутива из текстов программ (3→4) – это техническая задача, решаемая использованием компиляторов и прочих вспомогательных инструментальных программ.

Основное назначение Генератора, это автоматизация этапа 2→3, т.е. автоматизация процесса создания исходных текстов программ на основе описания проекта. Для этого был разработан специальный язык описания проектов, ориентированный на аналитика. Этот язык является тем формализмом, который позволяет осуществить переход 2→3 автоматически.

Язык описания проектов

Проект программного комплекса или системы представляет собой набор файлов, в которых содержится описание содержательных понятий и объектов разрабатываемой системы, логических структур данных, бизнес - процедур системы и пользовательского интерфейса. Основными объектами языка описания проекта являются:

- реквизиты проекта,
- платформы, для которых предполагается генерация программ,
- пользователи системы,
- типы данных,
- документы,
- сетевые структуры данных,
- реляционные базы данных,
- автоматически генерируемые SQL - запросы,
- произвольные SQL - запросы,
- прикладные серверы,
- порты прикладных запросов,
- WEB - порты серверов,
- бизнес - процедуры прикладных серверов,
- «ручные» программы бизнес-процедур,
- пользовательские окна,
- пользовательские диалоги,
- пользовательские приложения.

Реквизиты проекта задают имя проекта и другие выходные данные, связанные с разработкой проекта:

project <идентификатор проекта>:
"<Полное наименование проекта>";
/version=<номер версии>
/copyright="<информация о copyright>"
/author=<информация об авторах описания проекта>
/baseport=<начальный порт TCP/*IP по умолчанию>*
/language=<языки>

Платформой в разных контекстах принято называть разные вещи. Например, можно рассматривать разные аппаратные платформы – Intel, VAX, Alpha, Power PC и пр. Также можно рассматривать разные операционные системы – MS Windows, Linux, Free BSD, Open VMS и пр. С точки зрения баз данных можно под платформой понимать разные системы управления базами данных (СУБД) – MS SQL Server, Oracle, Sybase, MySql и пр. С точки зрения пользовательского интерфейса можно различать использование Win32 GDI, Motif, Gtk и пр. С точки зрения web-интерфейса можно различать web-сервера Apache, Microsoft IIS. И, наконец, с некоторой натяжкой можно к платформам отнести разные функциональности – сервер, ИНТЕРНЕТ-сервер, клиентское приложение.

В Генераторе процесс генерации предусматривает использование разных платформ. Тексты программ для каждой платформы могут быть сгенерированы независимо в разное время. В качестве основного языка реализации используется Си (не Си++). В зависимости от используемой аппаратной платформы и операционной системы могут использоваться разные компиляторы. В пределах одной платформы используется один компилятор языка Си. В Генераторе в настоящее время реализованы следующие платформы:

- Cltgtk – оконные приложения для использования в Linux и реализованные с помощью пакета gtk для X Windows.
- Cltgtw – аналог cltgtk, но для MS Windows.
- Cltwin – оконные приложения для использования в MS Windows.
- Libuni – библиотеки для ручных программ в Linux.
- Libwin – библиотеки для ручных программ в MS WIndows.
- Srvuni – серверы для использования в Linux.

- Srvwin – серверы для использования в MS Windows.

Отдельно могут быть заданы платформы по базам данных. При декларации базы данных необходимо указать набор драйверов, которые могут поддерживать интерфейс с конкретным типом СУБД. Указание типов драйверов СУБД должно быть среди реквизитов проекта:

/database=(< тип драйвера СУБД>[,<…>])

В настоящее время в Генераторе предусмотрены следующие драйверы:

- win_dblb7 – интерфейс с MS SQL Server через ntwdblib в MS WIndows.
- win_orcl – интерфейс с Oracle через oci32 в MS WIndows.
- win_sybase – интерфейс с SYBASE через ctlib в MS WIndows.
- win_mysql – интерфейс с MySql в MS Windows.
- uni_psql – интерфейс с Postgres в Linux.
- uni_sybase – интерфейс с SYBASE через ctlib в Linux.
- uni_mysql – интерфейс с MySql в Linux.

Следует отметить, что приведённые списки отражают текущее состояние Генератора, связанное с реально разрабатываемыми проектами.

Кроме возможности выполнения заданных SQL-запросов в драйверах баз данных предусматривается возможность анализа текущего состояния конкретной базы данных на предмет его соответствия требованиям текущей версии программного комплекса. Для этого в составе программ серверных платформ предусматриваются программы генерации sql-скриптов для создания новой пустой базы, а также для приведения существующей базы в состояние, требуемое для новых программ с сохранением содержимого. Это необходимо при смене версии работающей системы, когда программы новой очередной версии требуют изменения состава таблиц базы данных, при этом содержимое базы следует сохранить. Основная сложность в драйверах – именно эта возможность извлечь из существующей базы данных информацию о составе таблиц, полей, ключей, индексов и пр. в виде, пригодном для сравнения с видом, заложенным в проект. К сожалению, в этом вопросе разные СУБД демонстрируют

абсолютно различные средства, как по информативности, так и по удобству.

Пользователи генерируемой системы подразделяются на группы. Каждой группе пользователей присваиваются определённые права и функции, которые доступны пользователям данной группы. Структуры, связанные с пользователями системы описаны ниже в статье А.Н. Широкова "Обеспечение информационной безопасности в архитектуре клиент-сервер".

Типы данных в описании проекта используются в различных контекстах. Тип данных может задаваться как:

- предописанный тип данных (numb,char,date,money,…), переопределение ранее определённого прототипа,
- перечислимый тип (enum, radio, mask),
- структура (struct).

Каждое описание типа вводит уникальный идентификатор типа, программное представление, компоненты и ряд опций.

type <имя> : <программное представление>
[(<компоненты>)]
{<опции>..}

Программное представление может быть задано в форме структуры компонент, в форме перечислимого типа, ссылкой на прототип или ссылкой на предописанный тип. Компоненты типа используются для задания констант перечислимого типа, констант масочного типа, имён элементов структуры с заданием их типов. Опции типа задают особенности использования этого типа в разных контекстах. Например, для типа может быть задано наименование *(/title=<строка>),* которое по умолчанию будет использовано в элементах диалога, заголовках таблицы и пр. Для перечислимых типов может быть задано представление в диалогах и интернет-браузерах не в виде выпадающего списка (по умолчанию), а в виде набора радио-кнопок. Для типа с программным представлением double можно задать количество цифр после десятичной точки. Для текстового типа можно задать, что вводимая текстовая переменная является паролем, т.е. в диалогах и в интернет-браузерах текст должен быть скрыт при вводе. Также для текстового типа можно задать признак использования html-тега <textarea> вместо стандартного

<input type=text>. Структурный тип, как и в большинстве языков, представлен последовательностью именованных компонент, уникальных в пределах данного типа.

Понятие документа является одним из центральных в описании проекта. Документ – это описание типа данных, имеющего сложную внутреннюю структуру:

document <имя документа>[: <имя структурного типа>];
{ record <имя записи> [: < имя структурного типа >];}…
{ set <имя>[owner <владелец>] member <член набора>;}…

Имена документов уникальны в пределах проекта. С документом может быть связан структурный тип. Это означает, что в каждом экземпляре такого документа хранится ровно один экземпляр такой структуры. Если других компонент в документе не объявлено, то такой документ можно считать контейнером для хранения структур заданного типа.

В документе может быть декларировано произвольное количество именованных записей (record). Имена записей уникальны в пределах документа. С записью может быть связан структурный тип. Это означает, что в каждом экземпляре записи хранится ровно один экземпляр структуры. Здесь уместна аналогия со структурами реляционных баз данных: запись документа – это таблица базы данных. Соответственно в одном экземпляре документа может быть размещено произвольное количество экземпляров записей. В отличие от базы данных с записью можно не связывать никакую структуру, это не парадокс – такие записи могут использоваться в наборах для организации связей.

В документе может быть декларировано произвольное количество именованных наборов (set). Имена наборов уникальны в пределах документа. Набор описывает связь между записями документа типа “один ко многим”. Для этого с каждым набором связана запись – владелец набора и, отдельно, запись – член набора. Если рассматривать аналогию с реляционной базой данных, то набор – это пара ключей – Primary key и Foreign key, только здесь отсутствуют входящие в состав ключа колонки таблицы, связь между записями организуется на уровне хранения внутренними ссылками. Это свойство и обусловливает уместность применения записей без вложенных

в них структур – эти записи используются для связывания, в них хранятся только физические ссылки, напрямую не доступные из программы.

Каждому экземпляру записи, являющемуся владельцем некоторого набора, соответствует, по определению, экземпляр набора. Каждый экземпляр набора, кроме соответствующего экземпляра владельца, может содержать некоторую упорядоченную последовательность экземпляров записей типа члена набора (возможно, пустую).

Набор может быть последовательным и ключевым. В последнем случае задаётся ключ в виде последовательности компонент записи – члена набора, по которым в лексикографическом порядке упорядочиваются члены набора автоматически при включении в набор. В последовательном наборе порядок записей задаётся при их включении указанием номера позиции.

Можно также описывать, так называемые сингулярные наборы, у которых отсутствует запись владелец. Каждый сингулярный набор в документе представлен ровно одним экземпляром. В некотором смысле владельцем сингулярного набора можно рассматривать сам документ, вернее, экземпляр документа для экземпляра набора.

Описание документа автоматически предполагает наличие в языке описания проекта операторов манипулирования содержимым документа. К таким операторам, в частности, можно отнести следующие действия.

- Записать в заданный экземпляр документа структуру.
- Считать из заданного документа структуру.
- Создать экземпляр записи данного типа с указанием структуры, содержимое которой нужно разместить в записи.
- Удалить заданный экземпляр записи.
- Считать из заданного экземпляра записи структуру.
- Записать в заданный экземпляр записи структуру.
- Включить заданный экземпляр записи в экземпляр набора в заданную позицию.
- Найти по заданному номеру позиции экземпляр члена набора по экземпляру владельца.

- Перейти от заданного экземпляра члена набора к следующему/предыдущему.
- Найти экземпляр владельца по заданному экземпляру члена набора.
- Для заданного владельца ключевого набора и значения ключа найти соответствующий экземпляр члена набора.

Здесь приведён далеко не полный перечень действий–операторов, предусмотренных для документа. Наборы реализуются двусвязными списками. Поэтому время выполнения большинства операций не зависит от количества членов в наборе (в том числе, найти первый/последний член набора). Некоторые операции выполняются за время O(log(N)), где N – количество членов в наборе. И только немногие за время, пропорциональное N (поиск члена набора по номеру позиции).

Сетевые структуры данных. Модель документа есть не что иное, как хорошо забытая (и совершенно напрасно) модель сетевой базы данных по спецификации КОДАСИЛ. Похожесть на модель реляционной базы данных не в счёт, так как там, во-первых, primary/foreign key реализуются в виде ограничений целостности, проверяемых во время модификации, а не в виде физических ссылок. Во-вторых, работа с документом предполагает использование операторов перечисленного выше вида, а не использование SQL-запросов (может быть, в будущих версиях Генератора и это будет реализовано, если где-нибудь понадобится).

Описанное выше в общих чертах понятие документа является некоей абстракцией. На самом деле, в Генераторе предполагается, по крайней мере, три реализации этой абстракции.

Первая реализация – документ в памяти компьютера. Эта реализация предполагает использование таких документов для самых разнообразных целей, например, для организации протокола между клиентом и сервером. Спецификация каждого запроса сервера задаёт список входных документов и список выходных документов. Это означает, что клиентский модуль перед выполнением запроса должен создать по экземпляру документа из списка входных для данного запроса. Далее эти документы, созданные в памяти приложения, преобразуются в текстовое представление и по сетевому каналу свя-

зи передаются на сервер. На сервере принятый текст используется для восстановления клиентских документов в памяти сервера, после чего эти документы передаются на обработку программе, предусмотренной для обработки запроса. Эта программа строит экземпляры документов типов, заданных в спецификации запроса. По завершении программы обработки запроса построенные выходные документы преобразуются в текстовое представление и по сетевому каналу передаются в качестве ответа клиентскому приложению. В клиентском приложении принятый текст вновь используется для восстановления документа в памяти. Полученные таким образом документы в клиентском модуле используются для выдачи информации на экран. Другой пример использования документа в памяти – это просто некоторое хранилище структурированной информации в памяти, вроде как база данных на время выполнения программы.

Вторая реализация – хранение документа в совокупности бинарных файлов с прямым доступом. По сути дела, это есть не что иное, как реализация **сетевой базы данных** с соответствующим интерфейсом. В отличие от документов, хранящихся в памяти компьютера только во время работы программы, данное представление обеспечивает хранение информации произвольно долго, как и должно быть в базе данных. Реализация базы данных обеспечивает однопользовательский (однопрограммный) интерфейс. Преимущество такой базы заключается в значительно более высокой скорости обработки данных, не связанной с поиском по произвольным наборам реквизитов.

И, наконец, третья реализация – сетевой (по TCP/IP) интерфейс к одной из двух предыдущих реализаций. Наиболее реально используемый вариант – сетевой интерфейс к файловой реализации –это многопользовательский интерфейс к сетевой базе данных. Справедливости ради следует отметить, что сетевая реализация может несколько снизить производительность, так как специфика сетевых баз данных – большое количество быстрых операторов, а в реляционной базе – малое количество тяжеловесных запросов.

Структуры реляционных баз данных в проекте описываются в виде совокупности таблиц, индексов и связей между таблицами,

которые определяются, как и для сетевых данных с помощью задания наборов(set):

{ table <имя таблицы> : <имя структурного типа>
[(<первичный ключ таблицы>)];
[<опции таблицы>]}…
{index <имя индекса> on<имя таблицы> [unique](<индекс>)}…
{ set <имя набора> owner <имя таблицы-владельца> member
<имя таблицы – членов набора>(<ссылка на владельца>);}…

Столбцы (поля) таблицы соответствуют компонентам структурного типа, описывающего таблицу. Первичный ключ и индекс – это списки имён столбцов таблиц. Первичный ключ должен быть заявлен в списке уникальных индексов. В описании набора ссылка на владельца – это список имён столбцов таблиц – членов набора. Ссылка на владельца в наборе должна по типам данных совпадать с первичным ключом владельца набора.

Автоматически генерируемые SQL-запросы распространяются только на одну таблицу базы данных. Список автоматически генерируемых запросов задаётся в опциях к таблице:

[select <имя запроса>(<вход >):(<выход >);]…
[insert < имя запроса > (<вход>);]…
[update < имя запроса > (<вход>):(<изменяемые поля>);]…
[delete < имя запроса > (<вход>);]…
[cursor < имя запроса > (<вход>):(<выход>)/(<порядок>);]…

Здесь *<вход>* – это список полей, которые используются в предикате поиска (WHERE), а *<выход>* – выходные переменные запроса.

Произвольные SQL-запросы, которые могут быть описаны в проекте, представляют собой также как и автоматически генерируемые запросы и реализуются в виде функций с входными и выходными параметрами. Для описания этих запросов используется стандартный SQL. Описание произвольных SQL-запросов в проекте имеет следующий вид:

sql <имя запроса> (<вход>):[(<выход>)]
<тело запроса на SQL>;

В теле запроса входные переменные (<вход>) могут явно использоваться как host-переменные в Embedded SQL для C. Выходные пе-

ременные (<*выход*>) имеют смысл только для поисковых запросов (*select & cursor for select*).

Прикладные серверы реализуют бизнес-процедуры. С каждым сервером может быть связана одна или несколько баз данных реляционного или сетевого типа. На рис. 1. приведён пример серверной архитектуры и связи серверов с другими компонентами системы.

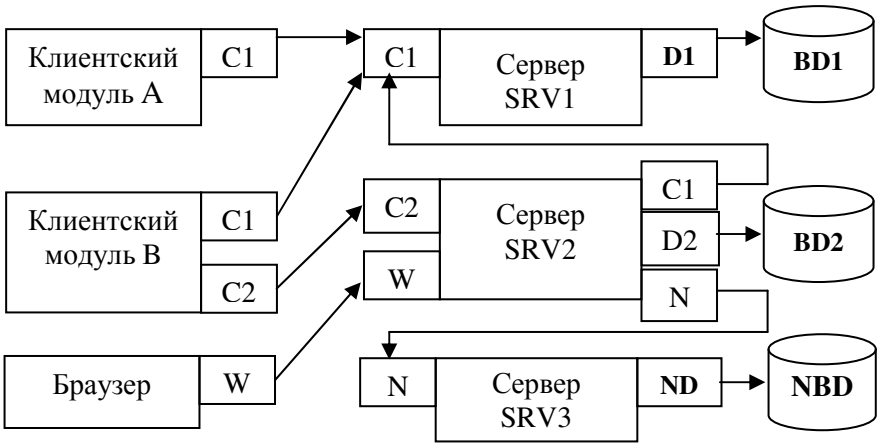


Рис.1. Пример клиент – серверной архитектуры

Сервер может иметь один или несколько **портов**, которые предназначены для связи с разными типами клиентских модулей. Через каждый порт сервера для пользователей доступны только те бизнес-процедуры этого сервера, которые приписаны к данному порту. Сервер может иметь несколько специализированных портов для исполнения запросов к сетевой базе данных. Сервер может иметь несколько портов, работающих по протоколу HTTP, для реализации WEB-интерфейса пользователей. И, наконец, сервер может иметь несколько клиентских интерфейсов для связи с другими серверами, по отношению к которым данный сервер выполняет роль клиента. На рис.1

- сервер SRV1 имеет порт C1 для приёма пользовательских запросов и интерфейс D1 с реляционной базой данных BD1;

- сервер SRV2 имеет порт C2 для приёма пользовательских запросов, порт W для приёма HTTP-запросов, клиентское соединение C1 с портом C1 сервера SRV1, интерфейс D2 с реляционной базой данных BD2, клиентское соединение N с портом N сервера SRV3 для запросов к сетевой базе данных;
- сервер SRV3 имеет порт N для приёма запросов к сетевой базе данных и интерфейс ND к сетевой базе данных NBD;
- клиентский модуль A имеет интерфейс C1 с портом C1 сервера SRV1 для передачи пользовательских запросов;
- клиентский модуль B имеет интерфейс C1 с портом C1 сервера SRV1 для передачи пользовательских запросов, интерфейс C2 с портом C2 сервера SRV2 для передачи пользовательских запросов;
- стандартный браузер имеет возможность передать запрос Интернет-порту W сервера SRV2.

Интернет-порт является специальным видом серверного порта, предназначенного для обеспечения санкционированного доступа к бизнес-процедурам серверов системы через Интернет. При описании интернет-порта задаётся состав именованных запросов с параметрами, которые обрабатывает этот порт. Для каждого такого запроса описывается программа реакции на него, в которой можно выполнять запросы к базам данных, другим серверам, которые в свою очередь будут обращаться к базам данных. При завершении работы каждой такой программы должно быть предусмотрено исполнение оператора, предписывающего создать и отправить в качестве ответа пользователю некоторую html-страницу. Для этого в файле предусмотрена возможность описания совокупности поименованных страниц с формальными параметрами. Описание каждой такой страницы представляет собой иерархическую структуру компонент специального вида, приблизительно соответствующих тегам html-языка. Для обеспечения большей гибкости в качестве одной из таких компонент можно использовать ссылку на один из ранее описанных блоков. Блоки – это аналоги страниц, но без html-заголовка. Интернет–порты сервера могут быть доступны для браузеров напрямую, либо через CGI-интерфейс через стандартный web-сервер (Apache, IIS).

Описание прикладного сервера имеет следующий вид:

server <имя сервера>;
[database <имя базы данных>:<тип реляционной базы данных>]…
[genbd <имя базы данных>:<тип реляционной базы данных>]…
{port <имя порта запросов>}…
{gbdport <имя порта сетевой базы>}…
{client <имя соединения> port <имя порта>}…
{web <имя web-порта>}…
{ <описание функций>}…
{<описание бизнес-процедур>}…

Функции, которые описаны в пределах прикладного сервера, носят служебный характер и используются при описании бизнес-процедур.

Бизнес-процедуры прикладных серверов – это функции сервера, подключённые к определённому порту:

request <имя порта>.<имя бизнес-процедуры>
input(<входные документы>)
output(<выходные документы>)
func { <тело бизнес-процедуры> }

Бизнес-процедуры принимают входные документы от пользователей, обрабатывают их, формируют ответ в виде выходных документов и передают их обратно пользователям. В теле функции бизнес-процедуры могут использоваться любые описанные выше функции, а также функции SQL-запросов используемых реляционных баз данных и операторы работы с сетевыми базами данных.

Пользовательские окна – это абстрактное понятие, связанное с внешним представлением описанных в проекте документов. В настоящее время в системе имеются предописанные оконные классы текстовых окон, таблиц и деревьев. На основе этих классов разработчик может описать именованные оконные типы, задав имена и типы переменных/документов в составе окна, а также задав программы, описывающие поведение окна в разных ситуациях.

window <имя окна> document <имя документа>
(<входные переменные>)
:(<выходные переменные>)
{tableview|textview|treeview}
[<описание компонент окна, а также управляющих команд >]

Квалифицированный разработчик имеет возможность описывать собственные оконные классы, задав для них программы на языке Си с соблюдением заданных интерфейсов. Эти классы могут быть использованы наравне с предописанными классами для задания оконных типов. Для реализации типовых табличных окон предусмотрен специальный макрос описания типовых окон.

Пользовательские диалоги представляют собой специальный вид окна, открытие которого блокирует доступ к окну приложения до тех пор, пока пользователь не закроет это окно.

dialog <имя диалога> document <имя документа>
(<входные переменные>)
:(<выходные переменные>)
[<описание компонент и команд диалога>]

Диалоги используются для ввода данных и для просмотра выводимой в диалог информации. В качестве компонент окна можно использовать разнообразные типовые элементы вроде окон ввода текста, выпадающих списков, птичек и пр. стандартных элементов. Кроме того, в окне можно использовать в качестве своих элементов описанные ранее окна. Размещение элементов окна управляется многочисленными опциями и специальными контейнерными элементами типа вертикальная коробочка, горизонтальная коробочка и пр.

Пользовательские приложения – это клиентские модули, обеспечивающие интерфейс пользователей с бизнес-процедурами серверов. Для приложения может быть объявлено, что оно является клиентом для заданного списка серверов.

application <имя пользовательского приложение>;<опции>
client <имя клиента> port <имя серверного порта>

С точки зрения пользовательского интерфейса в приложении декларируется список разных оконных видов – макетов, а также управляющие элементы для каждого макета.

layout <имя макета размещения окон> document <имя документа>
(<входные переменные>):(<выходные переменные>)
<геометрия размещения окон>
<описание управляющих команд и других свойств макета>

Макет определяет состав и взаимное расположение описанных выше окон внутри главного окна приложения. Управляющие команды определяют схему активизации окон макета, переход от макета к макету, вызов меню и диалогов, обращение к бизнес-процедурам сервера и тому подобное.

Кроме описанных выше основных понятий и объектов в Генераторе проектов есть ряд предописанных функций и макросов, предназначенных для описания проекта.

Все упомянутые выше понятия и объекты проекта упорядочены и представлены в различных файлах, совокупность которых составляет полное описание проекта.

Состав файлов описания проекта

Ниже приводится состав файлов, образующих в совокупности описание проекта, готовое для генерации программного кода разрабатываемой системы. Исполняемый модуль Генератора считывает информацию из перечисленных ниже файлов, производит синтаксический и семантический контроль, и, в случае успешного прохождения контроля, генерирует комплект исходных текстов программ вместе со скриптами для компиляции. Для получения дистрибутива разработчику достаточно запустить скрипты сборки проекта.

Головной файл проекта. Описание проекта может располагаться в нескольких файлах разных типов, но ключом ко всему проекту является головной файл описания проекта с расширением ".gen". В минимальном варианте описание проекта может состоять только из этого файла. В более сложных случаях в головном файле могут быть заявлены компоненты проекта, подробное описание которых располагается в отдельных файлах, которые обрабатываются Генератором после обработки головного файла в заданном порядке. Кроме задания структуры проекта в головном файле размещается описание типов, констант и документов, используемых в разных частях описания.

Файлы описания баз данных. В головном файле проекта могут быть перечислены типы баз данных с уникальными именами в пределах проекта. Для каждого такого типа базы данных предполагается наличие в описании проекта одноимённого файла с расширением ".dbs". Файл описания базы данных содержит описание состава

таблиц, индексов, связей, а также подробное описание именованных SQL-запросов с параметрами. Описанные таким образом базы данных могут быть использованы в других компонентах проекта.

Файлы описания серверов. В головном файле проекта могут быть перечислены серверы с уникальными именами в пределах проекта. Для каждого такого сервера предполагается наличие в описании проекта одноимённого файла с расширением ".srv". Файл описания сервера содержит подробную информацию о сервере: используемые базы данных, порты-интерфейсы для обработки клиентских запросов, спецификация и реализация запросов к серверу по всем портам и пр.

Файл описания типов окон. Если в проекте предполагается использовать оконный интерфейс, то можно указать в специальной опции, что в проекте присутствует файл описания окон, имеющий имя, совпадающее с именем проекта, и расширение ".win".

Файл описания диалогов. Если в проекте предполагается использовать оконный интерфейс, то можно указать в специальной опции, что в проекте присутствует файл описания диалогов, имеющий имя, совпадающее с именем проекта, и расширение ".dlg".

Файл описания приложений. В головном файле проекта могут быть перечислены приложения с уникальными именами в пределах проекта. Для каждого такого приложения предполагается наличие в описании проекта одноимённого файла с расширением ".app". Файл описания приложения содержит подробную информацию о приложении.

Файлы программ на языке Си. При описании некоторых компонент проекта может быть предусмотрено использование Си-программ, написанных вручную, которые необходимо включить в состав проекта. Для этого в синтаксисе описания предусмотрены соответствующие средства. Кроме того, для большинства таких программ предусматривается генерация заготовок, которые могут быть использованы разработчиком в качестве образца для написания реальной программы. Каждая такая заглушка при отсутствии реальной функциональности демонстрирует используемые интерфейсы и синтаксически готова для включения в состав проекта.

Генерация и сборка программного комплекса

Программа Генератора проектов реализована в виде консольного приложения для платформ Win32 и Linux. Выполняемый файл Генератора не зависит от платформы. Для генерации проекта необходимо запустить Генератор в рабочем каталоге описания проекта и указать в качестве параметра идентификатор проекта, а также набор опций для указания требуемых платформ. Для каждой указанной платформы генерируется программный код системы. При генерации из рабочего каталога описания проекта копируются специфицированные в головном файле проекта программные коды, написанные вручную. 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 автоматическая генерация отдельных модулей 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 предусматривают взаимодействие с сервером в соответствии с требованиями подсистемы безопасности.

Вместе с текстами программ для каждой указанной платформы генерируются скрипты для сборки программ для получения дистрибутивов системы.

С точки зрения состава программных компонент в дистрибутиве могут присутствовать программы следующего вида:

- серверы – программы, обрабатывающие запросы от клиентских программ на выполнение бизнес-процедур;
- клиентские модули с оконным интерфейсом;

- библиотеки программ, обеспечивающие выполнение бизнес - процедур;
- вспомогательные программы – конфигураторы баз данных, CGI-интерфейсы и пр.

Заключение

Описанная технология работы с Генератором проектов позволяет в значительной степени сократить затраты на программирование. В идеальном случае, если решаемая задача достаточно типовая, аналитик может обойтись без услуг программистов. Программисты могут понадобиться только в случае, если возникает необходимость использовать какие-либо нестандартные программные средства, не включённые в состав Генератора (например, программы работы с периферийным банковским оборудованием, что весьма актуально для финансовых приложений).

ОБЕСПЕЧЕНИЕ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ В АРХИТЕКТУРЕ КЛИЕНТ-СЕРВЕР

А.Н. Широков

Введение

Одной из важных задач при разработке финансовых систем является обеспечение информационной безопасности [10]. В связи с этим в рамках инструментального комплекса Генератор проектов этому вопросу уделяется особое внимание. Возможности этой инструментальной системы позволяют проектировать и создавать прикладные системы, отвечающие самым современным требованиям рынка программного обеспечения в области информационной безопасности. Эти требования включают в себя высокую гибкость, эффективность и функциональность средств информационной защиты при проектировании прикладных систем, построенных по технологии распределённого многоуровневого клиент-сервера. Эффективность обработки прикладных запросов с применением средств защиты информации в Генераторе проектов достигается благодаря возможности грамотного распределения нагрузки по различным компонентам системы.

Особую роль современные методы защиты информации играют в финансовых приложениях. Тенденции развития электронной коммерции, мобильного банкинга, Интернет - платежей диктуют необходимость уделять существенное внимание обеспечению информационной безопасности всех компонент прикладной системы. Особенно необходимо выделить такие задачи, как идентификацию и аутентификацию пользователей и клиентов системы, шифрование и дешифрование при передаче финансовой информации по публичным каналам.

В данной работе предлагается технология организации сетевого взаимодействия и обеспечения информационной безопасности в прикладных системах, базирующихся на технологии многоуровне-

вого клиент-сервера. Данная технология реализована в рамках Генератора проектов.

Архитектура системы

Инструментальная система позволяет проектировать и создавать прикладные системы в архитектуре многоуровневый клиент-сервер. Основными компонентами клиент - серверной архитектуры являются:

- Сервер Безопасности (crp).
- База Данных Сервера Безопасности (crp db).
- Прикладная Составляющая (app).
- Порт Сервера (prt).
- База Данных Прикладного Сервера (app db).
- Прикладной Клиентский Модуль (clt).
- Модуль Администратора Безопасности (adm).

Условно в любой конфигурации многоуровневого клиент-сервера можно рассмотреть три типовых уровня. Это уровень клиентских модулей или клиентских компонент, уровень прикладных серверов и уровень баз данных.

К уровню клиентских модулей относятся Прикладной Клиентский Модуль и Модуль Администратора Безопасности.

К уровню прикладных серверов относятся Сервер Безопасности, Прикладная Составляющая и Порт Сервера. Их объединение в одном процессе называется Прикладным Сервером.

Уровень баз данных состоит из Базы Данных Сервера Безопасности и Базы Данных Прикладного Сервера.

Порт Сервера в предлагаемой системе также может относиться одновременно и к клиентским модулям, если функционирует в роли удалённого Сервера Безопасности (об этом будет написано далее).

Ниже представлены различные конфигурации многоуровневых клиент-серверов, построенных в рамках предлагаемой модели.

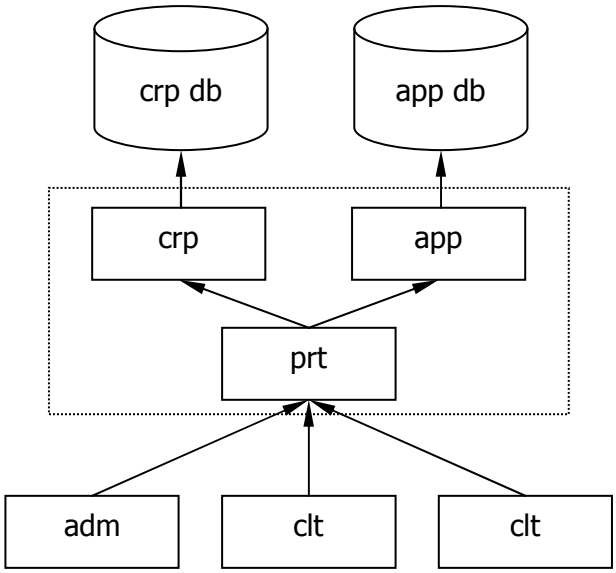


Рис. 1. Простейший трёхуровневый клиент-сервер

Схема, изображённая на рис.1, представляет собой простейший трёхуровневый клиент-сервер. Первый уровень представлен клиентскими модулями, куда входят Модуль Администратор Безопасности и Прикладные Клиентские Модули. Второй уровень представлен Прикладным Сервером, состоящим из Порта Сервера, Сервера Безопасности и Прикладной Составляющей. В третий уровень входят База Данных Сервера Безопасности и База Данных Прикладного Сервера.

Принцип работы такой схемы следующий. Клиентская сторона по сетевому каналу подключается к Порту Сервера. Производится процедура идентификации/аутентификации с Сервером Безопасности в соответствии с используемой схемой аутентификации. В результате успешной аутентификации вырабатываются сессионные ключи для данного сетевого подключения. Эти ключи используются для дальнейшего шифрования/дешифрования передаваемых данных.

После успешной аутентификации на клиентской стороне возможно формирование запросов. Сформированный запрос, передаётся по сетевому каналу на Прикладной Сервер через Порт Сервера.

Если запрос был сформирован Модулем Администратора Безопасности, то обработка производится в Сервере Безопасности. При необходимости происходит обращение к Базе Данных Сервера Безопасности. Сформированный ответ отправляется обратно клиентской стороне.

Если запрос был сформирован Прикладным Клиентским Модулем, то первоначально запрос дешифруется в Сервере Безопасности, затем обработка производится в Прикладной Составляющей. При необходимости происходит обращение к Базе Данных Прикладного Сервера. Сформированный ответ шифруется в Сервере Безопасности и отправляется обратно клиентской стороне.

Схема, изображённая на рис.2, представляет собой многоуровневый клиент-сервер с удалённым (выделенным) Сервером Безопасности.

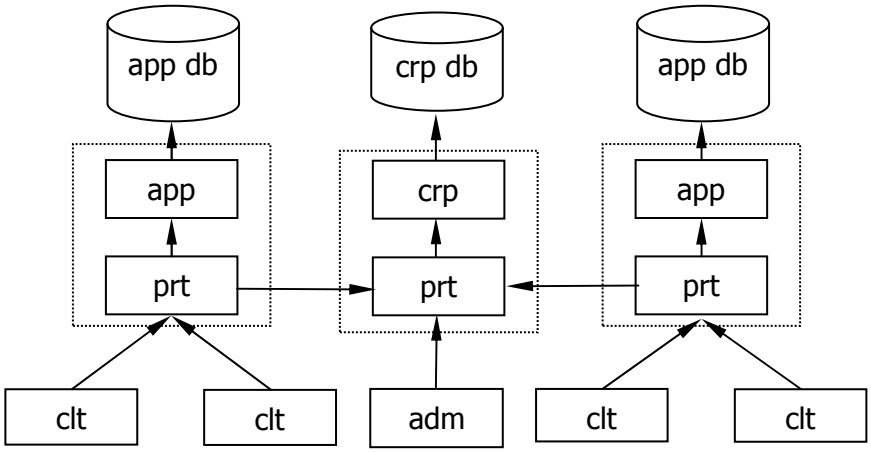


Рис. 2. Конфигурация с двумя Прикладными Серверами и одним удаленным Сервером Безопасности

Основные отличия такой схемы от простейшего трёхуровневого клиент-сервера состоят в следующем. Аутентификация клиентских модулей производится централизовано на одном Сервере Безопасности независимо от количества используемых Прикладных Серверов. Соответственно преимуществом такой схемы является единая база пользователей и привязанных к ним привилегий при использовании нескольких Прикладных Серверов. Для администрирования Сервера Безопасности единый Модуль Администратора Безопасности подключается непосредственно к Порту Сервера удалённого Сервера Безопасности. Это, несомненно, упрощает ведение базы пользователей и связанной с ними информации.

Шифрование и дешифрование данных производится либо на удалённом Сервере Безопасности, либо непосредственно в Портах Сервера Прикладных Серверов в зависимости от конфигурации.

Первый вариант является более защищённым в силу того, что сессионные ключи не покидают процесс удалённого Сервера Безопасности.

Второй вариант является более производительным в силу того, что нет необходимости дополнительно передавать зашифрованные и дешифрованные данные по сетевому каналу Порт Сервера – удалённый Сервер Безопасности.

Схема, изображённая на рис.3, представляет собой ещё более сложный и разветвлённый многоуровневый клиент-сервер. В этой конфигурации удалённый Сервер Безопасности совмещает в себе одновременно и функции прикладной обработки. Это может быть необходимо, например, для ведения статистического учёта работы пользователей, зарегистрированных на Сервере Безопасности. Соответственно к такому Серверу Безопасности могут непосредственно подключаться Прикладные Клиентские Модули для взаимодействия с Прикладной Составляющей.

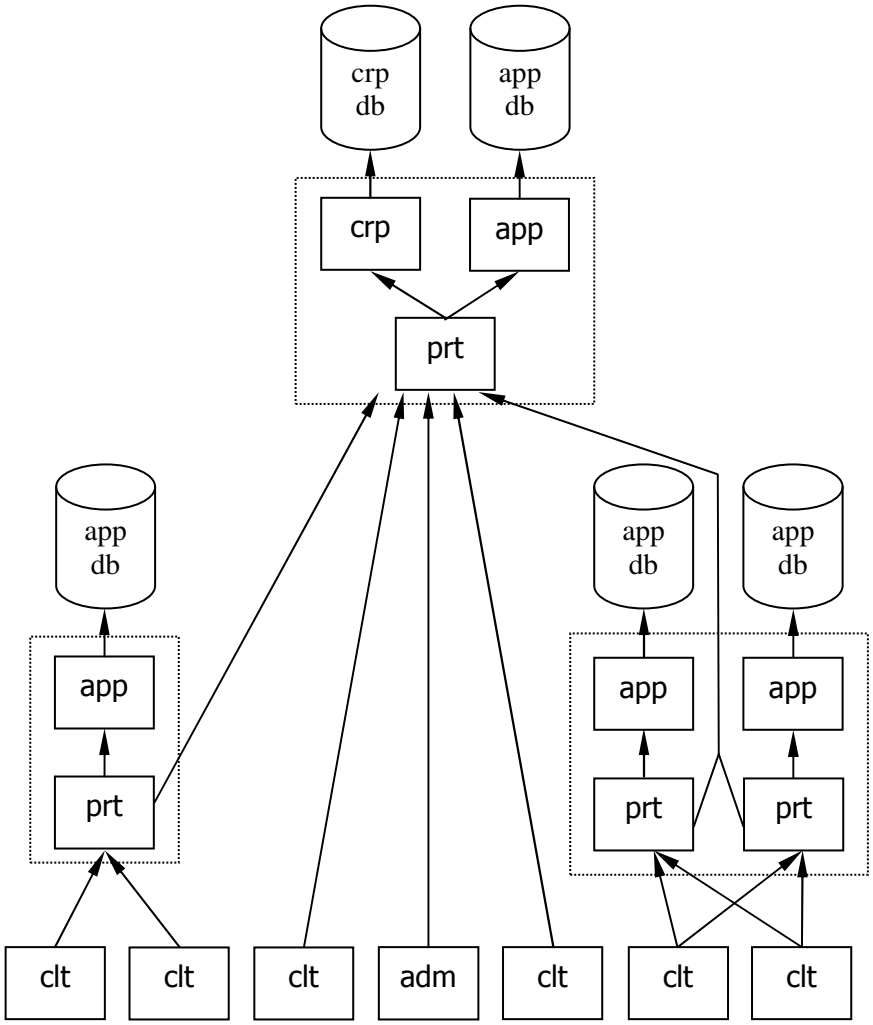


Рис. 3. Конфигурация, содержащая Прикладной Сервер с одним Портом, Прикладной Сервер с двумя Портами и Прикладной Сервер с одним Портом и локальным Сервером Безопасности

Помимо этого в приведённой конфигурации второй Прикладной Сервер содержит в себе два Порта Сервера, две Прикладных Составляющих и две Базы Данных Прикладного Сервера. Это в некотором смысле является самой общей конфигурацией непосредственно Прикладного Сервера. В такой конфигурации запросы, приходящие на каждый Порт Сервера обрабатываются независимо в соответствующей Прикладной Составляющей, каждая из которых взаимодействует со своей базой данных независимо.

Другие возможные конфигурации Прикладного Сервера допускают наличие нескольких Портов Сервера, к которым привязана одна Прикладная Составляющая, взаимодействующая с одной базой данных или несколькими базами данных. Такая конфигурация удобна для распределения нагрузки на работу Порта Сервера по обработке сетевого трафика. Эта ситуация может возникнуть, когда в системе преобладают запросы, которые быстро обрабатываются в Прикладной Составляющей. Тогда основная нагрузка ложится на Порт Сервера. Поэтому увеличение их количества позволяет распределить нагрузку.

Другая ситуация, в которой может потребоваться такая конфигурация, может возникнуть, если есть необходимость разделения доступа по сетевым интерфейсам, т.е. доступ из локальной сети позволяет выполнять один набор запросов, доступных только из Порта Сервера, работающего на локальном сетевом интерфейсе. А запросы, приходящие из глобальной сети, будут обрабатываться на Порте Сервера, работающем на глобальном сетевом интерфейсе. Такая конфигурация наиболее типична для интернет-магазинов, где по локальному сетевому интерфейсу производится обслуживание содержимого интернет-магазина, а по внешнему сетевому интерфейсу производится обслуживание клиентов интернет-магазина.

Также возможна конфигурация, в которой к каждому Порту Сервера привязана своя Прикладная Составляющая, но все они взаимодействуют с одной базой данных или с одними и теми же базами данных. Это может быть удобно в случае, если используются два сильно различающихся алгоритма прикладной обработки данных, но сами данные находятся в одной базе данных. Единственный недостаток такой схемы состоит в том, что необходимо на прикла-

ном уровне решать проблему одновременного обращения к одним и тем же записям в базах данных.

Программная реализация системы безопасности

С точки зрения операционной системы каждый Прикладной Сервер представляет собой отдельный процесс. Для обслуживания сетевых подключений в рамках одного процесса может быть создано произвольное количество экземпляров Порта Сервера. Каждый экземпляр Порта Сервера функционирует независимо один от другого и может обслуживать следующие типы сетевых подключений:

- Сетевые подключения от Прикладных Клиентских Модулей.
- Сетевые подключения от Модулей Администраторов Безопасности.
- Сетевые подключения от Прикладных Серверов.

В терминах операционной системы каждый Порт Сервера характеризуется наличием слушающего TCP/IP порта, задаваемого в параметрах Порта. Все экземпляры получают свою уникальную ссылку HPRT.

Порт Сервера работает только во взаимодействии с Сервером Безопасности. Сервер Безопасности выполняет функции генерации, хранения и распространения ключевой информации пользователей, функции идентификации/аутентификации пользователей, функции шифрования/дешифрования данных и проверки полномочий пользователей. В рамках одного процесса Прикладного Сервера может быть создано произвольное количество экземпляров Сервера Безопасности. Каждый экземпляр Сервера Безопасности взаимодействует с Базой Данных Сервера Безопасности. Каждый Порт Сервера должен работать в одном из двух режимов:

- Режим с локальным Сервером Безопасности.
- Режим с удалённым Сервером Безопасности.

Тип экземпляра Сервера Безопасности задаётся при его создании в параметрах конфигурации. Все экземпляры получают свою уникальную ссылку HCRP.

Локальный Сервер Безопасности целиком функционирует в рамках процесса Прикладного Сервера, в котором он создан. При создании локального Сервера Безопасности указывается привязка к соответствующей Базе Данных Сервера Безопасности.

Удалённый Сервер Безопасности в действительности представляет собой клиентскую компоненту, осуществляющую транспортные функции по передаче запросов через сетевой канал на другой Прикладной Сервер, в рамках которого функционирует локальный Сервер Безопасности. При создании удалённого Сервера Безопасности задаются параметры сетевого подключения и клиентская аутентификационная информация.

При создании Порта Сервера указывается ссылка HCRP на требуемый экземпляр Сервера Безопасности. В результате весь трафик, проходящий через этот Порт, будет обрабатываться на указанном Сервере Безопасности.

Для обслуживания сетевых подключений от Прикладных Клиентских Модулей к соответствующему Порту Сервера привязывается процедура прикладной обработки, которая при необходимости может взаимодействовать с Базой Данных Прикладного Сервера.

Для обслуживания сетевых подключений от Модулей Администраторов Безопасности к соответствующему Порту Сервера привязывается процедура обработки запросов, которая взаимодействует с Базой Данных Сервера Безопасности.

Порт Сервера может обслуживать сетевые подключения от Прикладных Серверов, которые будут обрабатываться на привязанном к этому Порту Сервере Безопасности. Такой Порт Сервера выступает в роли удалённого Сервера Безопасности.

Структура Базы Данных Сервера Безопасности

Ключевая информация пользователей, их привилегии и другие данные подсистемы безопасности хранятся в Базе Данных Сервера Безопасности. Схема базы данных представляет собой сетевую структуру, состоящую из различного типа записей и наборов. На рис. 4 приведена схема Базы Данных Сервера Безопасности.

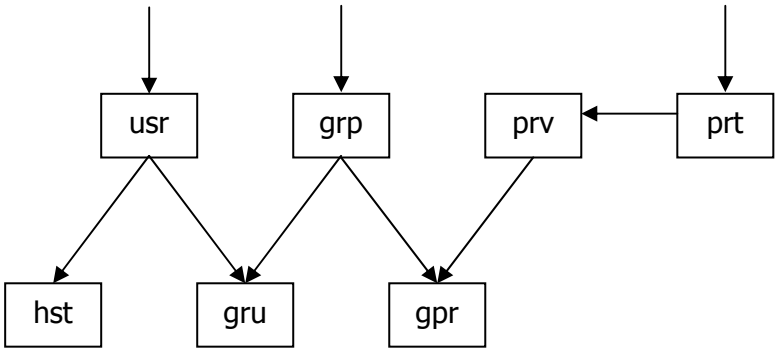


Рис. 4. Схема Базы Данных Сервера Безопасности

В базе данных присутствуют следующие типы записей:

- usr – пользователь,
- grp – группа,
- prv – привилегия,
- prt – порт,
- hst – необратимая свёртка пароля,
- gru – принадлежность пользователя к группе,
- gpr – принадлежность привилегии к группе.

В базе данных присутствуют следующие типы наборов:

- crp_usr – сингулярный набор пользователей,
- crp_grp – сингулярный набор групп,
- crp_prt – сингулярный набор портов,
- usr_hst – набор свёрток паролей пользователя,
- prt_prv – набор привилегий порта,
- usr_gru, grp_gru – наборы, обеспечивающие привязку пользователей к группам,
- grp_gpr, prv_gpr – наборы, обеспечивающие привязку привилегий к группам.

Тип записи usr содержит уникальное в рамках Сервера Безопасности имя пользователя, ключевую информацию пользователя и другую описательную информацию. Ключевая информация зависит от типа применяемой аутентификации. Для аутентификации на симметричных ключах ключевой информацией является необратимая

свёртка пароля пользователя. Для аутентификации с применением цифровых сертификатов ключевой информацией является различительное имя сертификата (Distinguished Name).

Тип записи grp содержит уникальный в рамках Сервера Безопасности идентификатор группы и её наименование.

Тип записи prv содержит уникальный в рамках Сервера Безопасности идентификатор привилегии и её наименование.

Тип записи prt содержит уникальный в рамках Сервера Безопасности идентификатор порта и его наименование.

Тип записи hst содержит необратимую свёртку пароля пользователя.

Тип записи gru является вспомогательным и указывает на принадлежность пользователя к группе.

Тип записи gpr является вспомогательным и указывает на принадлежность привилегии к группе.

Сингулярный набор crp_usr содержит список всех зарегистрированных на Сервере Безопасности пользователей.

Сингулярный набор crp_grp содержит список всех групп пользователей.

Сингулярный набор crp_prt содержит список всех зарегистрированных на Сервере Безопасности Портов Сервера.

Набор usr_hst содержит список последних необратимых свёрток пароля пользователя. Он служит для проверки ввода нового пароля при смене пароля пользователя. Количество хранимых свёрток определяется правилами безопасности Сервера Безопасности.

Набор prt_prv содержит список привилегий конкретного Порта Сервера. Этот набор формируется автоматически по следующему правилу. Порт Сервера в режиме удалённого Сервера Безопасности подключается к Порту Сервера, работающему с локальным Сервером Безопасности. При подключении на Порт Сервер Безопасности присылается список используемых на Прикладном Сервере привилегий. По идентификатору подключившегося Порта Сервера определяется имеющийся в базе данных список привилегий. Привилегии из присланного списка, которые отсутствуют в хранящемся списке, автоматически вносятся в него, но никому не присваиваются, а привилегии, которые есть в хранящемся списке, но отсутствуют в при-

сланном, удаляются из хранящегося списка и соответственно удаляются все ссылки на него. Таким образом обеспечивается безопасное обновление списка привилегий на Сервере Безопасности. Для назначения новых привилегий пользователям системы предназначен Модуль Администратора Безопасности.

Наборы usr_gru и grp_gru обеспечивают привязку пользователей к группам пользователей через вспомогательную запись gru. Любой пользователь может входить в любое количество групп, и в любой группе может содержаться любое количество пользователей.

Наборы grp_gpr и prv_gpr обеспечивают привязку привилегий к группам пользователей через вспомогательную запись gpr. Любая привилегия может принадлежать любому количеству групп, и к любой группе может быть привязано любое количество привилегий.

Таким образом, на основании информации, хранящейся в Базе Данных Сервера Безопасности, обеспечивается следующее.

- Ведётся список зарегистрированных пользователей и хранится их ключевая информация.
- Все пользователи разбиты на смысловые группы.
- Каждой группе привязан список доступных привилегий.
- Каждый пользователь обладает списком доступных привилегий на основании принадлежности пользователя к группе и привилегий к группе.
- Ведётся список зарегистрированных Портов Сервера.
- Для каждого Порта Сервера ведётся список используемых на нём привилегий.

Заключение

Разработанная инструментальная система безопасности в рамках Генератора Проектов обладает очень высоким уровнем гибкости и функциональности и при этом позволяет обеспечить высокую степень защищённости конфиденциальной информации от несанкционированного доступа. Учитывая все современные тенденции развития информационной безопасности [11], в инструментальной системе применены самые современные алгоритмы.

ГЕНЕРАЦИЯ ПРОГРАММНОГО КОДА В ГЕНЕРАТОРЕ ПРОЕКТОВ

Е.Н. Широкова

Введение

Основной функцией инструментального комплекса Генератора проектов является построение на основании исходного описания проекта программного кода разрабатываемых прикладных систем на языке С. Одной из важных задач развития Генератора проектов является проблема оптимизации и упрощения текста программ на заключительной стадии генерации файлов. При генерации программ постоянно существует противоречие между сложностью алгоритмов генерации и длиной генерируемого текста, структурой генерируемых программ. Для получения компактных, хорошо читаемых, прозрачных программ приходится разрабатывать весьма сложные алгоритмы генерации. Не всегда эту проблему удаётся решить, и тогда приходится мириться с порой неоправданным усложнением программного кода. Для преодоления этого недостатка автоматической генерации программ необходима разработка более совершенных моделей представления генерируемых программ, которые позволяли бы существенным образом оптимизировать их структуру и текст.

Генерация программного кода в Генераторе проектов

Процесс построения программного кода осуществляется в несколько этапов. Генератор проектов проводит синтаксический и семантический анализ, преобразует текст в некоторое внутренне представление, а затем по этому представлению генерирует программные коды.

Существующий алгоритм генерации программных кодов представляет собой процесс последовательного создания набора файлов программных кодов на языке С и ряд вспомогательных файлов, необходимых для дальнейшей сборки и сопровождения системы. Каждый файл формируется построчно (точнее, пооператорно). Для последовательного формирования файла используется набор специальных внутренних функций Генератора проектов. Такие функции

аналогичны стандартным функциям форматированного вывода, но дополнены строгой проверкой соответствия количества полей, подставляемых в результирующую строку, шаблону вывода. Генератор проектов проверяет соответствие заявленного для функции количества полей количеству ссылок на поля в шаблоне вывода.

Общая проблема такого последовательного алгоритма генерации состоит в том, что синтаксис языка С накладывает определённые ограничения на последовательность написания тех или иных элементов программного кода (объявление переменных, вызов функций, включение заголовочных файлов), которая не всегда совпадает с логикой исходного представления проекта и логикой работы анализатора описания проекта. Для пояснения уровня возникающих проблем при генерации программного кода приведём несколько примеров.

В процессе формировании тела функции (когда блок объявления переменных уже сформирован, а также сформирована часть тела функции) зачастую встаёт задача объявления внутри функции вспомогательной переменной (например, переменная цикла). При условии строго последовательного построчного формирования в Генераторе проектов тела функции из параметризованных фрагментов кода приходится усложнять алгоритм формирования программного кода. Программный код генерируется дважды, сначала генерируется «черновик», а потом проводится повторный анализ и при необходимости добавляется объявление вспомогательной переменной. Однако такого излишнего усложнения алгоритмов можно избежать путём разделения формирования программных кодов на два этапа:этап создания модели программного кода и этап построения файлов программных кодов по модели. При этом дополнительные переменные функции можно объявлять по мере возникновения необходимости, т.е. просто добавляя в модель информацию о новой переменной, которую нужно объявить в теле функции.

Ещё одним примером подобного рода проблем является случай рекурсивного вызова функции в пределах одного файла. Вызов функции рекурсивен, когда функция вызывается (используется) в файле ранее, чем описано её тело. Для преодоления данного противоречия в языке С необходимо описать (ранее вызова функции) про-

тотип функции. Однако при последовательной (построчной) генерации программных кодов заранее нельзя определить необходимость описания прототипа функции до её использования и описания, поэтому алгоритм генерации усложняется повторным анализом сгенерированного программного кода на предмет выявления необходимости формирования прототипов. При создании модели программы нет необходимости отслеживать необходимость описания прототипа, а на этапе генерации программных кодов по модели прототип рекурсивно вызываемой функции формируется автоматически.

В качестве ещё одного аналогичного примера можно указать необходимость определять в начале формирования С-файла, какие h-файлы нужно включать в программу. При использовании этапа предварительного создания модели программы такая проблема не возникает. В процессе формирования модели программы при необходимости вызова в с-файле функций, описанных в некотором h-файле, этот заголовочный файл включается в модель и затем уже автоматически включается в генерируемый С-файл.

Частью исходного описания проекта могут быть функции на языке С, которые написаны «вручную» и которые должны быть включены в программное обеспечение разрабатываемой системы. При генерации программного кода системы файлы с этими функциями просто копируются в соответствующие каталоги и интегрируются в общем программном коде. Однако в «ручных» программах вероятность содержательных ошибок существенно больше, чем в программах, автоматически генерируемых в рамках единой информационной модели. Кроме того, стиль разработчика «ручных» программ (имена переменных, функций, общее оформление программного кода, наличие комментариев и т.д.) может существенно отличаться от «стиля» автоматически сгенерированных программ. В связи с этим, на наш взгляд, целесообразно разработать специальную экспертную систему, которая бы умела анализировать программный код «ручных» функций, строить модели проанализированного программного кода и сигнализировать о найденных противоречиях в этих моделях. Используя полученную модель, программный код может быть сгенерирован заново, повторяя имеющуюся функцио-

нальность (если нет ошибок), но будучи оформленным в соответствии с общим единым стилем, принятым в Генераторе проектов

Структурная модель генерируемых программ

Для совершенствования процесса формирования программных кодов в рамках Генератора проектов необходимо разработать инструментальную подсистему манипулирования структурой больших программ. Для создания такой подсистемы необходимо разработать модель генерируемых программ, которая бы включала:

- набор базовых элементов,
- допустимые связи между базовыми элементами,
- правила построения из базовых элементов и связей между ними результирующего программного кода.

Модели программ должны отвечать определённым требованиям. Модель должна быть достаточно полной для задач Генератора проектов. Здесь не ставится задача создать инструмент формирования произвольных конструкций языка С. Необходимо создать инструментарий, поддерживающий только те конструкции, которые необходимые для Генератора проектов (впрочем, надо сказать, что это множество конструкций весьма представительно).

Разрабатываемая модель должна учитывать возможность и удобство её использования для создания экспертной системы анализа программных кодов с целью последующей регенерации в принятом в Генераторе проектов стиле программирования. В рамках этой работы должны быть реализованы следующие операции с моделью:

- построение модели путём вызова специфицированных функций программного интерфейса подсистемы,
- преобразование модели в файлы программных кодов,
- построение модели экспертной системой по файлам программных кодов.

Поскольку структура данных модели во многом определяет интерфейс манипулирования этими данными, необходимо предусмотреть удобство построения модели. Применение функций конструирования модели программных кодов должно быть более эффективным, чем непосредственное формирование программного кода из типовых параметризованных фрагментов программных кодов на

языке С. Последовательность и логика создания базовых элементов программного кода и связей между ними должна определяться логикой работы Генератора проектов, а не особенностями языковых конструкций языка С.

Создание подсистемы манипулирования структурой больших программных комплексов предполагает решение следующих задач:

- определение базовых элементов используемых в Генераторе проектов языковых конструкций С и связей между ними,
- разработка формальной модели программных кодов, состоящей из базовых элементов и допустимых связей между ними,
- создание алгоритма формирования комплекта файлов программных кодов по модели программных кодов,
- разработка программного интерфейса подсистемы манипулирования структурой больших программных комплексов в части построения модели,
- создание алгоритма анализа программных кодов для экспертной системы с целью построения модели программных кодов.

Заключение

Одной из важных задач по развитию Генератора проектов является задача оптимизации заключительного этапа генерации программных кодов с целью упрощения генерируемых программных кодов прикладных систем. В данной статье обсуждается постановка задачи такой оптимизации. На наш взгляд, эта задача может быть решена путём разделения процесса генерации программных кодов на этап создания модели программы и этап генерации по модели файлов программных кодов. Для этого функции формирования программных кодов выделяются в отдельную подсистему манипулирования структурой больших программных комплексов. Для разработки такой подсистемы необходимо:

- определить адекватную модель программных кодов с уровнем абстракции выше, чем используемый язык программирования (язык С), и достаточно полную для целей Генератора проектов.
- определить программный интерфейс для построения модели.
- разработать формальный алгоритм генерации программных кодов по модели.

- разработать формальный алгоритм анализа программных кодов и построения по ним модели программных кодов.

Эти задачи являются предметом ближайшего развития проектного подхода при разработке прикладных информационных систем.

ПРОЕКТ БАНКОВСКОЙ СИСТЕМЫ

Л.Л. Вышинский

Введение

Современные системы автоматизации банковской деятельности являются одними из наиболее сложных и трудоёмких программных комплексов. Создание столь сложных и критических приложений невозможно без использования современных инструментальных и технологических средств проектирования, разработки и сопровождения. Именно для автоматизации проектирования банковских систем и был разработан инструментальный комплекс Генератор проектов.

Современные автоматизированные банковские системы должны удовлетворять очень многим требованиям, которые диктуются не только сиюминутными потребностями тех или иных подразделений банка, но и общей стратегией развития банковского бизнеса. Сейчас уже прошло время бурного неупорядоченного роста этой сферы, когда информационные технологии только осваивались в банковском деле. Прошло время, когда банки были весьма неразборчивы по отношению к внедряемым у них автоматизированным системам, когда ставилось множество разнообразных, зачастую несовместимых программных продуктов, решающих отдельные локальные задачи. В условиях роста глобализации экономики, укрупнения банковских структур, бурного развития телекоммуникационных сетей и компьютерных технологий перед банковскими автоматизированными системами встают проблемы постоянного повышения оперативности и качества обслуживания клиентов, доступности и прозрачности банковских технологий, максимальной интеграции вычислительных и информационных ресурсов, гибкости и мобильности, способности к быстрой адаптации к новым условиям. В связи с этим к автоматизированным банковским системам должны предъявляться новые ранее не выдвигаемые требования, связанные с качеством математических моделей, положенных в основу разрабатываемых проектов. Модели должны быть доступны для понимания и в достаточной степени

формализованы. Основной идеей, заложенной в Генераторе проектов, является создание в проекте формального описания системы, с тем чтобы этот проект являлся неотъемлемой обязательной компонентой внедряемой банковской системы, и все дальнейшие изменения в системе проходили бы через модификацию проекта. В данной статье излагается концепция и описывается архитектура проекта новой современной банковской системы. Разумеется, в рамках статьи нет возможности полностью представить проект банковской системы, но его основа, так сказать, аванпроект может и должна обсуждаться.

Основные концепции проекта банковской системы

1. Создание единого информационного пространства банка – основная цель внедрения автоматизированной банковской системы.
2. Единая архитектурная реализация функций банка – это минимизация затрат и максимизация порядка в информационных технологиях.
3. Разумное сочетание централизации и децентрализации банковских процессов – основа эффективности функционирования банковской системы.
4. Гибкие технологические схемы функционирования банковской системы – залог её долговечности и выживаемости в условиях изменчивости правил и норм банковской среды.
5. Реализация автоматизированной технологии ведения финансовых проектов – основа планирования в банке.
6. Финансовый анализ и прогнозирование в банковской системе – это перспектива развития банковского бизнеса.

Структура данных банковского проекта

Во главе описания проекта лежит структура данных. Эти данные должны адекватно отражать ту систему понятий, которая лежит в основе проекта, и те задачи, которые предполагается в этом проекте решать. На рис. 1 представлена логическая схема данных проекта банковской системы.

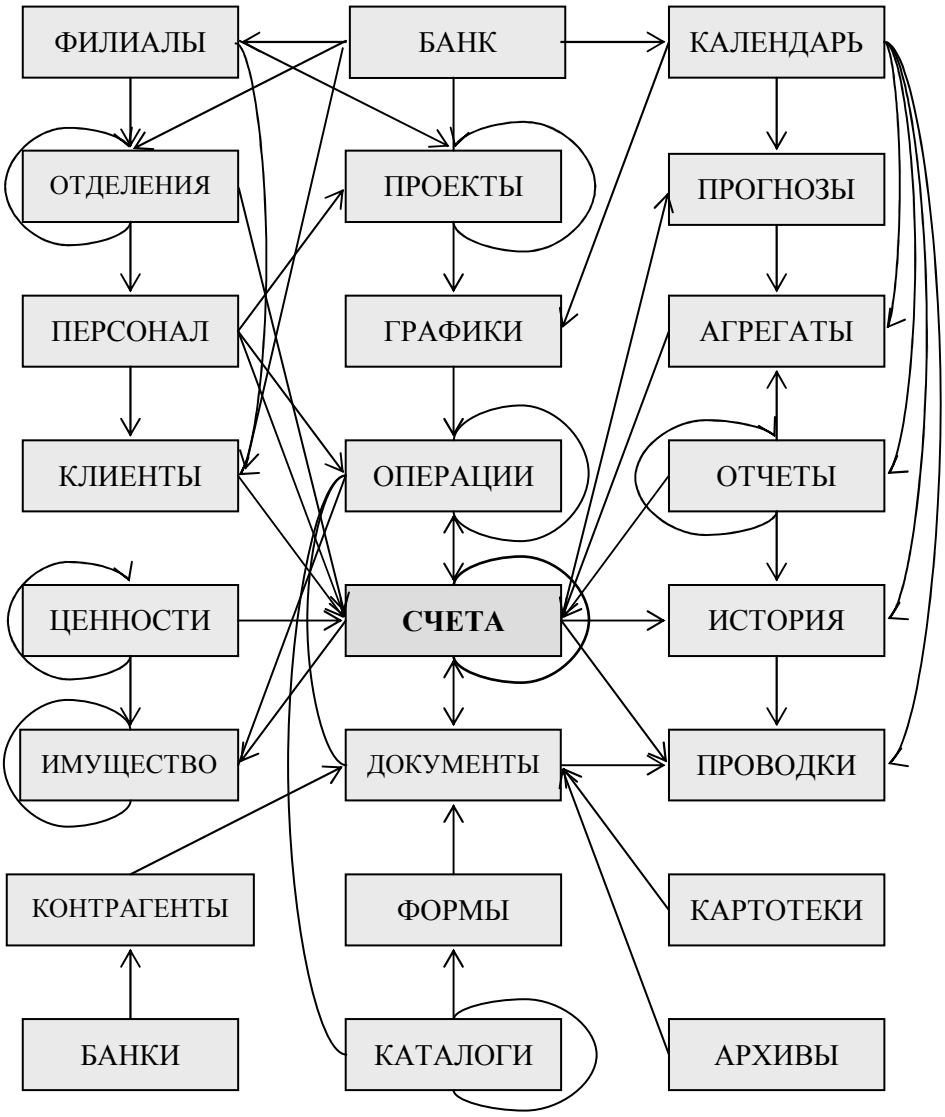


Рис. 1. Логическая схема данных проекта

На рис.1 прямоугольниками отражены основные содержательные объекты и понятия (реализуемые в виде таблиц баз данных), с которыми должна работать система, а дугами – отношения и логические связи между ними. Основные понятия, с которыми должна оперировать банковская система, можно объединить в смысловые группы.

Организационные структуры банка. К этой группе данных относятся следующие объекты.

БАНК – это вершина всей организационной структуры, соответствующая головному банковскому учреждению, в котором установлена система. В этой таблице должна быть ровно одна запись, в поля которой заносится вся необходимая для работы информация о банке.

ФИЛИАЛЫ – подчинённые головному банковскому учреждению самостоятельные филиалы или другие банковские учреждения, являющиеся подчинёнными, но обладающие юридической самостоятельностью. Заметим, что филиал банка может выступать как головное банковское учреждение в рамках данной банковской системы, если в этом филиале установлен отдельный её экземпляр.

ОТДЕЛЕНИЯ – любые подразделения банка или его филиалов, которые не являются самостоятельными банковскими учреждениями. Связь этой таблицы, образующая петлю, означает, что подразделения банка могут быть организованы в иерархические структуры.

ПЕРСОНАЛ – сотрудники банка. Сотрудники банка входят в структурные подразделения, могут иметь определённый статус, обязанности, уровень доступа к системе и к определённым блокам информации. Сотрудники, которые имеют доступ к системе, должны быть зарегистрированы средствами подсистемы безопасности.

КАЛЕНДАРЬ организационное средство, предназначенное для обеспечения деятельности банка в соответствии с дисциплиной операционных периодов. Операционные периоды в банковском учёте и отчётности играют первостепенную роль. Ведение внутреннего календаря может предусматривать различные операционные периоды – операционные дни, пятидневки, недели, месяцы, кварталы, годы, период заключительных оборотов и другие периоды, которые предусмотрены правилами ЦБ РФ и внутренней дисциплиной банка.

Дисциплина операционных периодов предусматривает их открытие и закрытие. Все операции привязаны к определённым операционным периодам. В базе данных хранится информация о текущих и прошедших периодах. Для целей планирования своей деятельности база данных системы может содержать информацию о будущих операционных периодах.

Партнёры и клиенты банка. К этой группе данных относятся КЛИЕНТЫ банка, КОНТРАГЕНТЫ по финансовым операциям и сторонние БАНКИ. Последняя группа данных может быть связана с одной или несколькими расчётными системами, в рамках которых функционирует наш БАНК .

Ценности образуют отдельную группу данных, в которых содержится перечень видов финансовых инструментов, которые используются в банке. Основным финансовым инструментом являются деньги. Все операции, все объекты финансовых и материальных отношений, с которыми имеют дело банковские системы, исчисляются в деньгах, в определённой валюте. Кроме валюты к ценностям относятся драгоценности, если они хранятся в банке, а также ценные бумаги. Отношение подчинённости, обозначенное на схеме, относится к классификатору, который разбивает всё множество финансовых инструментов на классы и группы. Кроме классификатора с каждым видом ценности связаны таблицы цен, котировок, курсов, которые регулярно обновляются в соответствии с принятыми правилами.

Учётные инструменты. Основным механизмом бухгалтерского учёта являются СЧЕТА. Ведение счетов достаточно жёстко регламентировано правилами бухгалтерского учёта. Эти правила выстраивают иерархию синтетических и аналитических счетов, задают способы их нумерации, определяют допустимые операции по этим счётам и ограничивают их применение в зависимости от их финансового состояния. СЧЕТА связаны с филиалами, подразделениями, ответственными исполнителями (персоналом), видом финансового инструмента (ценности), могут быть связаны с клиентами. Каждый счёт характеризуется своим финансовым состоянием, точнее, финансовым состоянием в текущие открытые операционные периоды (входящие и исходящие остатки, дебетовые и кредитовые обороты).

Кроме того, для каждого счёта хранится ИСТОРИЯ его финансового состояния за прошедшие операционные периоды и выполненные по этим счетам операции – ПРОВОДКИ. Учёт ИМУЩЕСТВА банка ведётся как в денежном, так и в натуральном исчислении. Стоимость имущества, его переоценка, амортизационные отчисления и другие финансовые характеристики ведутся на разных счетах рублёвого баланса банка. В натуральном исчислении имущество ведётся на инвентарных карточках, в которых указываются наименование по паспорту, инвентарный номер, место установки и/или эксплуатации, текущее состояние и другие характеристики. Всё имущество классифицируется и кодифицируется в соответствии с действующими классификаторами.

Финансовые и другие документы. Основанием для выполнения любой финансовой операции является финансовый или платёжный документ. Для выполнения операций в базе данных должна быть создана и сохранена электронная копия необходимого финансового документа. Электронные документы в системе создаются различным образом: путём ввода информации с бумажного оригинала, путём ввода и обработки определённых входящих финансовых сообщений, путём создания электронного оригинала документа с последующей распечаткой его в бумажном виде. Вопрос согласования «бумажного» и «электронного» документооборота должен решаться на основании действующих правил.

В данном проекте выдвигается концепция полностью документированной работы банковской системы. Это означает, что любая модификация информации в базе данных, ввод новых записей в содержательные таблицы должен быть подтверждён электронной копией документа, на основании которого выполнена эта операция. Открытие нового счёта, модификация его реквизитов, регистрация нового клиента, ввод нового сотрудника в базу данных, изменение банковской структуры – всё должно быть документировано.

Электронный документ может иметь различные статусы – статус подготовки, согласования, подписания, утверждения, исполнения, сторнирования, архивирования и так далее. Документы могут перемещаться по КАРТОТЕКАМ ответственных исполнителей и по

картотекам, предусмотренным технологией выполнения отдельных операций.

Банковские проекты. Деятельность банка и его учреждений ведётся в самых различных сферах, включая основные операции и хозяйственные операции по обеспечению основной деятельности. Как правило, задачи, решаемые банками, могут быть разбиты по различным направлениям, а в некоторых случаях могут быть сформулированы конкретные цели или ожидаемые результаты этой деятельности. Для целенаправленного планирования деятельности банка, для консолидации информации о доходах и расходах в процессе достижения поставленных целей, в банковской системе вводится понятие банковского ПРОЕКТА, которое определяется как некоторый механизм идентификации деятельности банка в достижении некоторого фиксированного результата. По сути, проект объединяет в одно целое совокупность в той или иной степени связанных между собой финансовых ОПЕРАЦИЙ, которые выполнятся в рамках заданных временных ГРАФИКОВ.

Аналитические инструменты. Деятельность банка, выполнение им своих основных функций невозможна без тщательного анализа результатов и оценки эффективности принятия тех или иных решений при проведении финансовых операций. Для реализации полноценного финансового анализа деятельности банка в автоматизированной системе необходимо существенное внимание уделять созданию специальных аналитических инструментов. Наиболее очевидными инструментами финансового анализа являются различные финансовые ОТЧЕТЫ, в том числе и стандартные банковские отчётные формы (балансы, оборотные ведомости, …). Однако для полноценного анализа одних отчётов недостаточно. Необходимы другие аналитические инструменты, в частности различные обобщённые показатели – АГРЕГАТЫ, директивные нормативы, различные коэффициенты, которые вычисляются по формулам на основании состояния и истории счетов. Важным результатом аналитической работы является составление ПРОГНОЗОВ. По каждому счёту, по каждому финансовому агрегату на основании информации, хранящейся в базе данных, может быть составлен его прогноз в любой заявленный будущий операционный период. Методы прогноза со-

стояния счетов и агрегатов в будущие операционные периоды могут быть самыми разными – от простейших усреднений «исторического опыта» до сложных и тонких методов прогнозирования по всей совокупности данных. Прогноз может вестись в режиме реального времени и использоваться в задачах планирования, управлениями активами и пассивами, в выборе тактики и стратегии развития банка.

Банковские технологии. Функционирование современной банковской системы невозможно без управления банковскими технологиями. Основными элементами банковских технологий являются:

- ведение КАТАЛОГА банковских операций,
- ведение каталога ФОРМ платёжных документов,
- построение и модификация структуры банковских счетов, финансовых агрегатов, прочих инструментов бухгалтерского учёта,
- построение и модификация свойств картотек документов,
- построение и модификация схем проводок и маршрутизации платёжных документов,
- построение и модификация схем начисления процентов на депозитные, кредитные счета и т.д.

Приведённая логическая схема данных проектируемой банковской системы не исчерпывает все необходимые типы данных и виды используемых объектов. Здесь приведены лишь основные объекты и понятия. Существуют ещё некоторые вспомогательные данные, которые необходимы для детализации конкретных банковских технологий. Они, естественно, должны появиться в полном описании проекта системы.

Функциональные модули банковской системы

Функциональные модули банковской системы должны обеспечивать работу всех групп ответственных исполнителей. Ниже дан перечень этих модулей и групп модулей, объединённых в подсистемы.

1. Модуль администратора системы.
2. Модуль руководителя банка.
3. Модуль главного бухгалтера.

4. Модуль бухгалтера / контролёра.
5. Модуль контролёра / кассира.
6. Модуль расчётно-кассового обслуживания.
7. Модуль обслуживания по вкладным операциям.
8. Модуль обслуживания по кредитным операциям.
9. Модуль межбанковских расчётов.
10. Модуль по расчётам с дебиторами и кредиторами.
11. Модуль по операциям с ценными бумагами.
12. Подсистема хозяйственной деятельности:
    - модуль управления имуществом,
    - модуль ведения хозяйственных договоров,
    - модуль начисления зарплаты.
13. Подсистема биллинга:
    - модуль администратора биллинга,
    - модуль приёма платежей,
    - модуль самообслуживания.
14. Мобильный банк.
15. ИНТЕРНЕТ – банк.
16. Модуль финансового анализа.
17. Модуль планирования и бюджетирования.
18. Модуль управления портфелем банка.
19. Модуль контрольно-ревизионной службы.
20. Модуль банковского технолога.

Архитектура системы

В основе архитектуры банковской системы лежит клиент – серверная технология. На рис.2 представлена структурная схема системы.

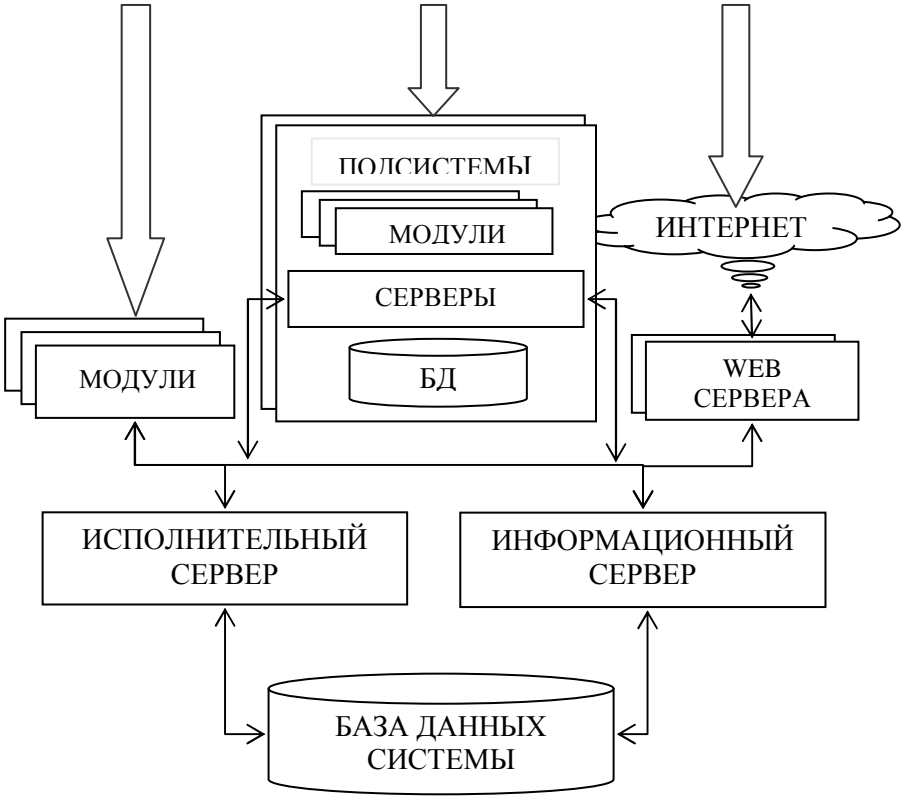


Рис. 2. Архитектура системы

Основными компонентами системы являются база данных, работающая под управлением СУБД, прикладные сервера, клиентские модули, WEB-серверы и подсистемы. Программный смысл этих компонент был изложен в предыдущих статьях данного сборника. Подсистемы устроены совершенно так же, как сама система. У подсистемы есть своя база данных, свои прикладные серверы и свои клиентские модули. Подсистема может взаимодействовать с прикладными серверами головной системы и через них получать и мо-

дифицировать информацию центральной базы данных. В свою очередь центральный сервер может обращаться к серверам подсистем и иметь доступ к информации их баз данных. В рамках подсистем могут быть реализованы те функции, которые могут выполняться автономно без тесного взаимодействия с центральной базой данных, лишь изредка обмениваясь относительно небольшими объёмами данных. Выше были приведены две функциональные подсистемы – подсистема хозяйственных операций и подсистема биллинга. Основной функцией биллинговой подсистемы является регистрация задолженностей клиентов перед поставщиками коммунальных и других услуг, а также регистрация платежей клиентов в счёт погашения этих задолженностей. (Система биллинга рассмотрена в одной из статей этого сборника.) Очевидно, что информация о задолженностях потребителей коммунальных услуг является автономной и не связана с системой банковского учёта. Поэтому лицевые счета клиентов обрабатываются в биллинге, там же учитываются их платежи, а в банковскую систему передаются суммарные платёжные документы на перевод средств поставщикам услуг и комиссионных отчислений для банка.

Другим примером реализации подсистем могут быть экземпляры той же банковской системы, установленные в филиалах головного банка. В этих филиальных подсистемах ведётся самостоятельный балансовый учёт операций филиала, а в головную систему передаётся необходимая информация для построения сводных документов по всему банку.

Такая архитектура достигает сразу несколько целей, которые были заявлены в концепциях банковской системы:

- создаётся единое информационное пространство, где существует принципиальная возможность получить информацию из любой базы данных,
- реализуется разумная децентрализация функций системы,
- унифицирована архитектура и общесистемная организация информационных технологий в банке.

Отметим ещё одну особенность приведённой архитектуры. В серверной части системы выделены два типа серверов исполнительный сервер и информационные серверы. Таким разделением

серверных функций подчёркивается и достигается принцип полной документированности выполнения любых операций в системе. Пользователи системы обращаются к информационным серверам с запросами по поиску нужной информации. При этом они подготавливают, согласуют и утверждают необходимые документы для тех или иных банковских операций. Когда документы готовы, они посылаются на исполнительный сервер для их реализации и внесения изменений в соответствующие разделы базы данных. Такое разделение функций серверных компонент позволит повысить эффективность обработки данных и надёжность системы.

Кроме обычных клиентских модулей в приведённой архитектуре реализован механизм работы с системой через ИНТЕРНЕТ. Это достигается при помощи специальных WEB - серверов, которые связываются с прикладными серверами. Реализация подобной технологии в нашей системе позволяет включить в неё такие клиентские функции, как мобильный банк (работа со своими банковскими счетами с помощью мобильного телефона) и ИНТЕРНЕТ - банк (работа со своими банковскими счетами через ИНТЕРНЕТ)

Представленный здесь аванпроект банковской системы планируется реализовать с помощью инструментального комплекса Генератор проектов.

АВТОМАТИЗИРОВАННАЯ СИСТЕМА БИЛЛИНГА

Л.Л. Вышинский, Е.Н. Широкова

Автоматизированная система биллинга – это информационно-учётная система, ориентированная на обеспечение взаиморасчётов между поставщиками и потребителями услуг. Поставщики услуг выставляют счета на оплату предоставленных коммунальных и других услуг потребителям – клиентам биллинговой системы. Эти счета, попадая в биллинговую систему, регистрируются на реестрах потребителей как их задолженности перед поставщиками. Для погашения задолженностей потребители производят платежи, биллинговая система регистрирует эти платежи, корректирует реестр задолженностей и пересылает через банк документы для зачисления поступивших средств на счета поставщиков услуг.

Биллинговые системы, по сути, являются посредниками между населением, обслуживающими предприятиями и банками. Применение биллинга позволяет упростить и автоматизировать процедуры приёма и обработки платежей населения путём предварительного сбора информации о задолженностях клиентов и актуализации её в момент совершения платежа. С помощью биллинга банки автоматизируют важный сектор своей работы – платежи населения, избавляются от ручного ввода платёжных документов, поставщики услуг оперативно информируются о погашении клиентами своих задолженностей, а клиенты биллинга получают современный удобный уровень обслуживания. Биллинговая система может использоваться как для автоматизации работы фронтофисных систем, так и для обеспечения приёма платежей с помощью банкоматов, инфокиосков, электронных карт и прочих устройств самообслуживания.

Информационная модель биллинга

Главной задачей биллинговой системы является регистрация платежей населения и передача информации об этих платежах между участниками расчётов. Участниками расчётов являются:

- потребители услуг – плательщики,
- поставщики услуг – получатели платежей,

- филиалы банковских учреждений по приёму платежей,
- средства самообслуживания по приёму платежей,
- расчётные подразделения банков, обеспечивающие трансферт средств от плательщика к получателю.

Общая схема выполнения платежей с участием биллинга представлена на рис. 1.

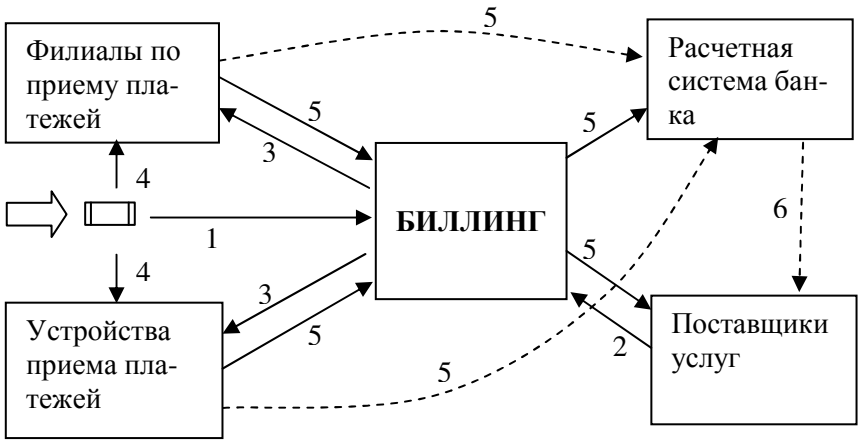


Рис. 1. Схема выполнения платежей: *1 – регистрация плательщиков в биллинге, 2 – счета от поставщиков на оплату услуг плательщиками, 3 – информация о задолженностях клиентов биллинга, 4 – погашение задолженностей клиентами биллинга, 5 – информация (или документы) по выполненным платежам, 6 – извещение о поступлении средств поставщикам услуг*

Для реализации этой общей схемы необходимо описание основных информационных объектов биллинга и их взаимодействия в процессе функционирования системы.

Обозначим через S множество предоставляемых услуг, которые зарегистрированы в системе биллинга:

$$S = \{s_i :< code, description, form, property >\}_{i=1,\ldots} \quad .$$

С каждым видом предоставляемой услуги связаны код, её краткое описание, форма платёжного документа и ряд свойств, определяющих правила работы с данной услугой в биллинге. Форма платёжного документа – это некоторый набор содержательных полей, описывающих параметры предоставленных услуг, и реквизиты для перечисления средств в счёт их оплаты. Поля и реквизиты формы могут быть связаны между собой определёнными соотношениями, которые либо проверяются при формировании (вводе) платёжных документов, либо используются для вычисления отдельных реквизитов.

С каждым видом услуг может быть связан один или несколько поставщиков этого вида услуги – они же получатели платежей. В биллинге должен вестись полный реестр получателей платежей, который здесь будет обозначен через P. В этом реестре, кроме названия получателя и вида оказываемой услуги, должны быть заданы его физический и электронный адрес, банковские и другие реквизиты для обеспечения передачи необходимой информации:

$$P = \{ p_j :< s^j, name, adres, email, bank, bank\_account, ... > \}_{j=1,...} .$$

Здесь $s^j \in S$. Получатель платежей для своих постоянных клиентов ведёт учёт предоставленных услуг и их оплату на лицевых счетах клиентов. В связи с этим в биллинге ведётся адекватный учёт задолженностей клиентов биллинга перед поставщиками услуг. Основным инструментом учёта в биллинге является реестр лицевых счетов потребителей услуг. Этот реестр ведётся в разрезе получателей платежей, т.е. для каждого получателя $p_i \in P$ ведётся свой реестр A_i:

$$A_i = \{ a_{i,j} :< p_i, account, name, adres, ..., debit, credit > \}_{j=1,...} .$$

Здесь $account$ – номер лицевого счёта. Кроме реквизитов владельца лицевого счёта (имя, адрес, телефон,…) в реестре фиксируется его текущее финансовое состояние: $debit$ - текущая задолженность потребителя перед поставщиком p_i и $credit$ - положительный остаток на счёте клиента при кредитовой форме предостав-

ления услуг или сумма переплаты клиентом, возникшая при погашении задолженностей. Финансовое состояние лицевого счёта представлено в сальдированной форме:

$$debit \geq 0, credit \geq 0; debit \times credit = 0 .$$

Финансовое состояние лицевого счёта клиента в биллинге формируется на основании следующей информации:

- сообщений, поступающих от поставщика услуг, о текущем финансовом состоянии счёта (задолженности либо положительном остатке) на фиксированную дату;
- предъявленных поставщиком счетов за оказанные услуги;
- информации об оплате клиентом предъявленных счетов за оказанные услуги;
- информации об авансовых платежах клиента в адрес поставщика услуг.

По всем этим видам входящей информации ведутся журналы регистрации. Журналы упорядочены по получателям платежей и по лицевым счетам:

а) журнал сообщений поставщика о состоянии лицевого счёта $a_{i,j} \in A_i$ на начало дня указанной даты

$$Jmess_{i,j} = \{mess_{i,j,k} :< a_{i,j}, date, debit, credit >\}_{k=1,...} ;$$

б) журнал счетов на оплату услуг, поступивших от поставщика по лицевому счёту $a_{i,j} \in A_i$

$$Jbill_{i,j} = \{bill_{i,j,k} :< a_{i,j}, date, sum, df >\}_{k=1,...} ,$$

где $date$ – это дата предъявления счёта, а sum - сумма, предъявленная к оплате по данному счёту; основанием для предъявления этой суммы к оплате является платёжный документ df , составленный в форме, которая соответствует виду предоставленных услуг;

в) журнал платежей, т.е. оплаты по предъявленным поставщиками счётам, или авансовых платежей

$$Jpay_{i,j} = \{ pay_{i,j,k} :< a_{i,j}, bill_{i,j,k}, date^*, sum^*, df^* > \}_{k=1,\ldots},$$

где $date^*$, sum^* и df^* в общем случае могут отличаться от соответствующих реквизитов в предъявленном счёте $bill_{i,j,k}$.

Финансовое состояние лицевых счетов и записи в журналах связаны следующими соотношениями для всех *i* и *j:*

$$\begin{aligned} & a_{i,j}.credit - a_{i,j}.debit \\ & = mess_{i,j,k_{i,j}}.credit - mess_{i,j,k_{i,j}}.debit \\ & - \sum_{k \in Kb_{i,j}} bill_{i,j,k}.sum + \sum_{k \in Kp_{i,j}} pay_{i,j,k}.sum . \end{aligned}$$

Здесь индекс $k_{i,j}$ соответствует последнему пришедшему сообщению о финансовом состоянии счёта $a_{i,j}$

$$mess_{i,j,k_{i,j}}.date = \max_k (mess_{i,j,k}.date),$$

а множества индексов $\mathrm{Kb}_{i,j}$, $\mathrm{Kp}_{i,j}$ по которым суммируются предъявленные счета и платежи определяются из условий

$$Kb_{i,j} = \{k : bill_{i,j,k}.date >= mess_{i,j,k_{i,j}}.date\},$$

$$Kp_{i,j} = \{k : pay_{i,j,k}.date >= mess_{i,j,k_{i,j}}.date\};$$

Владелец лицевого счёта и плательщик, который осуществляет погашение задолженностей или выполняет авансовые платежи, не обязательно является одним и тем же физическим (юридическим) лицом. Более того, один и тот же плательщик может обслуживать несколько лицевых счетов по разным получателям и разным видам услуг. В системе биллинга в качестве его постоянных клиентов регистрируются плательщики, то есть те, кто выполняет платежи по одному или нескольким лицевым счетам

$$C = \{c_i :< id, name, adres, ..., Ac >\}_{i=1,...} .$$

Здесь id - уникальный идентификатор плательщика, который регистрируется в биллинге и по которому он опознаётся. В качестве идентификатора может быть использован номер электронной платёжной карточки, с помощью которой плательщик может осуществлять платежи через биллинг в режиме самообслуживания. $Ac \subset \bigcup A_i$ – это множество лицевых счетов, по которым зарегистрированный плательщик может осуществлять платежи.

Платежи, выполняемые в системе биллинга, проходят через устройства приёма платежей, которые либо работают под управлением операторов платежей, либо обеспечивают плательщикам режим самообслуживания. Регистрация устройств приёма платежей в системе биллинга необходима для того, чтобы замкнуть цикл подготовки документов бухгалтерского учёта выполняемых платежей. Устройства приёма платежей регистрируются в биллинге в следующем виде:

$$Dev = \{dev_i :< numb, type, bank, account, ... >\}_{i=1,...} .$$

С каждым устройством кроме его уникального номера и типа связаны определённые банковские реквизиты, которые используются при построении бухгалтерских документов. Устройства приёма платежей, как правило, принадлежат банковским учреждениям, которые при необходимости могут быть зарегистрированы в биллинге в реестре получателей платежей в качестве посредника при оказании услуг. Мы выделим их как подмножество $Pb \subset P$.

Как уже говорилось выше, с каждым видом услуг, платежи по которым регистрируются в системе биллинга, связаны клиентские платёжные документы определённой формы. Однако банковское исполнение клиентских платежей сопровождается целым рядом операций, связанных с отчислениями комиссионного сбора, штрафов, пени, проводкой сумм по счетам устройств ввода, по счетам электронных карточек и прочее. Каждая из этих операций должна быть обеспечена соответствующими банковскими документами. Система биллинга осуществляет автоматическую подготовку необ-

ходимых документов в электронном виде и передачу их в АБС банка. Для автоматизации подготовки банковских платёжных документов в биллинге с каждым видом платежа определяется свой набор шаблонов необходимых выпускаемых документов как функции от реквизитов платёжной формы, от реквизитов получателя, от реквизитов плательщика и от реквизитов устройства ввода платежа:

$$T_i(p,c,d) = \{t_{i,j} :\prec s_i, type, fpayer(...), frecipient(...), fs(...),... \succ\}_{j=1,..}$$

Здесь $s_i \in S$ i-ый вид услуги. В функциях *fpayer, frecipient, fs*, определяющих шаблон, заданы способы вычисления реквизитов формируемых документов – плательщика, получателя, суммы, назначение суммы и прочее. Сами же документы формируются при выполнении платежа, передаются в банк и сохраняются в архиве биллинга:

$$Doc = \{d_n :\prec pay, doc :\prec type, payer, recepient, s,.. \succ\succ\}_{n=1,....}$$

Здесь $pay \in \bigcup Jpay_{i,j}$, а *doc*- документ, представленный в той форме, которая предписана в соответствующем шаблоне.

Таким образом, основными информационными объектами системы биллинг являются:

- S – множество видов услуг,
- F – множество форм клиентских платёжных документов,
- P – реестр получателей платежей, в том числе Pb – банки осуществляющие приём платежей,
- A – реестры лицевых счетов потребителей услуг,
- C – реестр плательщиков по счетам потребителей услуг,
- *Jmess* – журналы сообщений о состоянии лицевых счетов,
- *Jbill* – журналы счетов на оплату предоставленных услуг,
- *Jpay* – журналы выполненных платежей,
- *Dev* – журнал устройств приёма платежей,
- T – шаблоны банковских платёжных документов,
- *Doc* – архив банковских платёжных документов, подготовленных в системе биллинга для АБС.

На рис. 2 изображена сетевая информационная модель биллинга.

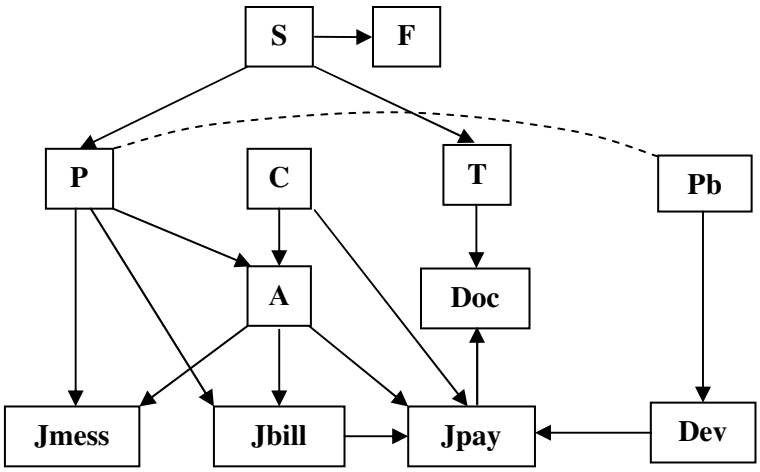


Рис. 2. Информационная модель биллинга

Основные функции системы

Для обеспечения функционирования описанной модели биллинга необходимо выполнение следующих основных функций:

- регистрация и описание услуг, которые можно оплатить с использованием системы биллинга;
- описание форм клиентских платёжных документов, которые используются при оплате услуг;
- регистрация и ведение реестра поставщиков услуг (получателей платежей);
- регистрация и ведение реестра лицевых счетов потребителей услуг;
- регистрация и ведение реестра плательщиков по счетам потребителей услуг;
- импорт информации от поставщиков услуг о текущем состоянии лицевых счетов потребителей услуг;
- импорт информации от поставщиков услуг о задолженностях плательщиков;
- ведение реестров задолженностей по каждому потребителю услуг;

- информационное обеспечение выполнения платежей (предоставление суммы общей задолженности плательщика, списка и реквизитов предъявленных счетов, другой информации);
- регистрация выполненных платежей и корректировка задолженностей плательщиков;
- вычисление сумм комиссионных отчислений по выполненным платежам;
- подготовка банковских платёжных документов по выполненным платежам для передачи в АБС банка (в том числе и документов по комиссионным отчислениям);
- подготовка и передача реестров выполненных платежей получателям;
- ведение журнала подключённых к биллингу устройств приёма платежей и их финансовый мониторинг.

Архитектура биллинга

Базовой архитектурой для биллинга является архитектура «трёхуровневый клиент-сервер».

На рис. 3 приведена структура и основные компоненты биллинговой системы, её окружение и информационные связи между программными модулями.

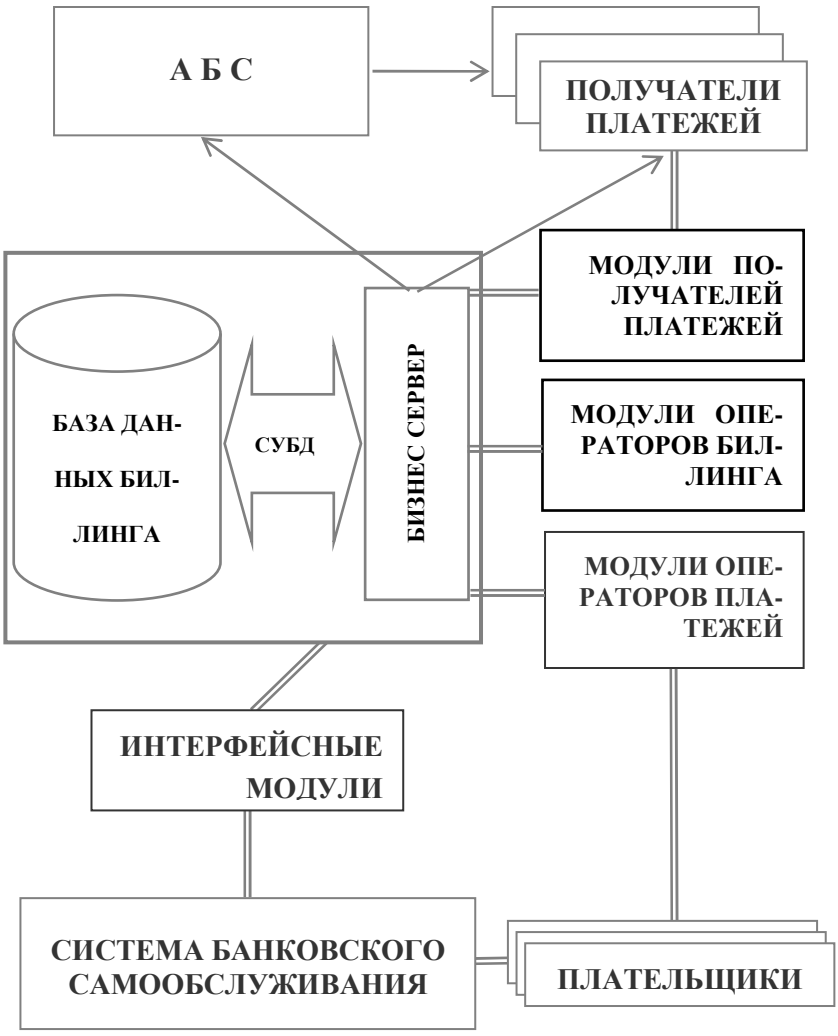


Рис. 3. Архитектура биллинговой системы

В составе биллинговой системы основными компонентами являются программный сервер биллинга (бизнес-сервер) и клиентские модули – модули операторов биллинга, модули операторов платежей, модули получателей платежей и специальные интерфейсные модули, которые необходимы для взаимодействия с банкоматами и инфокиосками системы самообслуживания (о системе самообслуживания см. [2]). Со всеми клиентскими модулями бизнес-сервер взаимодействует в режиме on-line, в том числе и с системой самообслуживания.

Функции операторов биллинга реализованы в трёх клиентских модулях:

- в модуле администратора биллинга,
- в модуле контролёра биллинга,
- и в модуле оператора отчётов.

Модуль администратора биллинга предназначен для ведения нормативно-справочной информации, а также для регулирования правил функционирования системы. Фактически эти операции сводятся к ведению и мониторингу информации, хранящейся в БД биллинга.

Модуль контролёра биллинга предназначен для ведения лицевых счетов потребителей услуг, справочника плательщиков, ввода, обновления и контроля информации по текущим задолженностям и выполненным платежам. Модуль ориентирован на схему работы с загрузкой от внешних систем текущей задолженности по лицевым счетам.

Модуль оператора отчётов биллинга предназначен для подготовки и вывода ежедневных отчётов по проведённым в биллинге платежам как для получателей платежей, так и для банков, которые осуществляют приём платежей.

Модули операторов платежей фактически являются фронтофисными подсистемами биллинга. С помощью этих модулей оператор может принимать платежи от клиентов наличными деньгами, проводя их через кассу, или в безналичном виде с использованием электронных пластиковых карт.

Модули получателей платежей предоставляют возможность поставщикам услуг оперативно получать информацию об оплате предоставленных услуг, о задолженностях своих клиентов и поставлять в биллинг информацию о предоставленных услугах, требующих оплаты.

Система биллинга может также взаимодействовать в режиме off-line с автоматизированными банковскими системами (АБС) и с поставщиками услуг – получателями платежей

Реализация системы биллинга

Система биллинга реализована с помощью Генератора проектов (Project Generator 2000 v. 3).

Объём проекта составил около 450 килобайт.

Сгенерированный программный код всей системы – около 7 мегабайт, в том числе клиентские модули около 4 мегабайт, различные библиотеки ~ 1 мегабайта, а бизнес-сервер – 2 мегабайта. Таким образом, эффективность Генератора проектов на системе биллинга оказалась достаточно высока, а соотношение ручного текста и текста полученного автоматически составило порядка 14.

СИСТЕМА БАНКОВСКОГО САМООБСЛУЖИВАНИЯ

И.Л. Гринёв, А.А. Логинов, Н.И. Широков, А.Н. Широков

Введение

Рассматриваемая в данной статье система банковского самообслуживания (далее Система) предназначена для обеспечения приёма безналичных платежей по банковским карточкам плательщиков через устройства банковского самообслуживания (банкоматы, инфокиоски и пр., далее – УС) и передачи информации о выполненных платежах в специальные биллинговые центры (см. предыдущую статью) в режиме реального времени. Основной концепцией Системы является обеспечение «интеллектуального интерфейса» между различными биллинговыми системами и программным обеспечением УС.

Основной проблемой, возникающей при организации приёма платежей от населения, является многообразие видов платежей и форм используемых платёжных документов. Это многообразие входит в противоречие со стандартным программным обеспечением УС, которое имеет весьма ограниченные возможности взаимодействия со своими пользователями. Поэтому возникает необходимость создания специального программного обеспечения, которое позволило бы организовать гибкую настройку и управление различными сценариями взаимодействия программного обеспечения УС с клиентами при выполнении платежей. Такую гибкую настройку сценариев работы с УС мы и называем здесь «интеллектуальным интерфейсом».

Архитектура Системы

Внешними программными комплексами, с которыми взаимодействует Система, являются: автоматизированные банковские системы (АБС), различные биллинговые центры поставщиков услуг, муниципальные расчётно-информационные центры, банковские подсистемы финансового обслуживания клиентов и т.п. На рис. 1 представлена архитектура Системы в рамках концепции «трёхуровневый клиент-сервер».

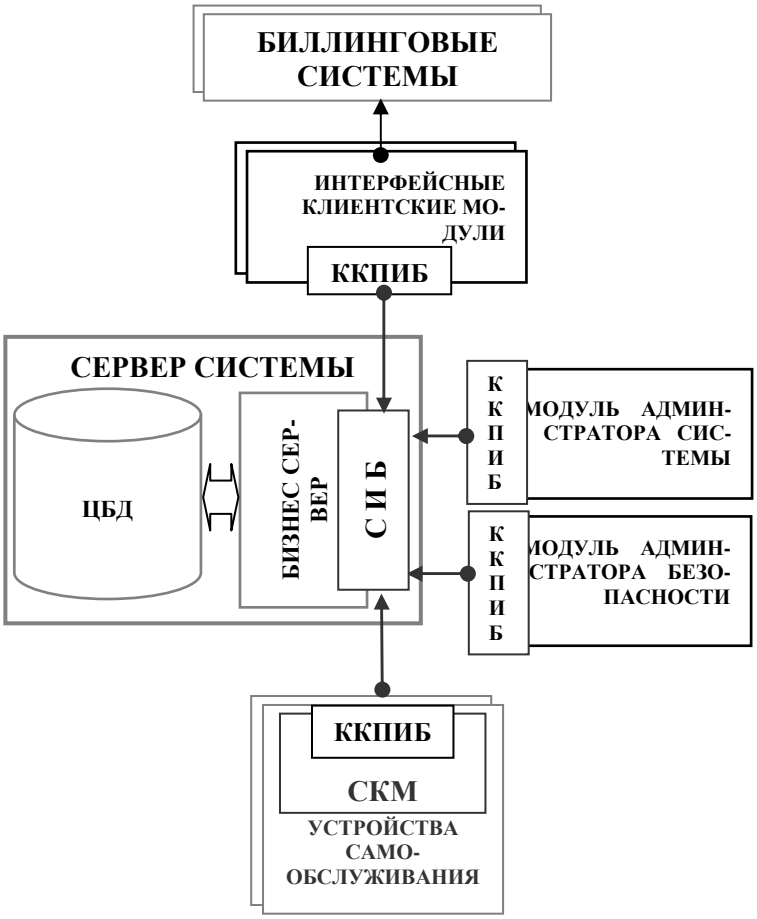


Рис.1. Архитектура Системы

В состав сервера Системы входят бизнес-сервер, взаимодействующий через СУБД с центральной базой данных (ЦБД), и сервер информационной безопасности (СИБ).

В состав клиентской части входят:

- модуль «Администратор Системы»;

- модуль «Администратор безопасности»;
- специализированные клиентские модули (СКМ), интегрируемые в программное обеспечение УС;
- интерфейсные клиентские модули (ИКМ) для связи с биллинговыми системами.

На рис.1 как составная часть клиентских модулей показаны клиентские компоненты подсистемы информационной безопасности (ККПИБ). Тот факт, что большинство стрелок соединяет ККПИБ и СИБ, следует трактовать так, что именно эти компоненты непосредственно взаимодействуют в процессе информационных обменов в Системе.

Модуль «Администратор Системы» предназначен для регулирования правил функционирования Системы. Фактически эти операции сводятся к ведению и мониторингу информации, хранящейся в ЦБД. Модуль «Администратор Системы» выполняет следующие функции:

- ведение списка биллинговых систем, подключённых к Системе;
- ведение списка УС, подключённых к Системе;
- просмотр информации по техническим платёжным операциям, электронным платёжным документам и их реквизитам;
- обработка «сбойных» платежей;
- просмотр информации по текущей активности клиентов, работающих на УС.

Модуль «Администратор безопасности» выполняет весь набор функций, связанный с обеспечением режима информационной безопасности.

Модули СКМ и ИКМ обеспечивают связь Системы с внешними программными комплексами. В Системе предусмотрена возможность одновременного взаимодействия с несколькими внешними биллинговыми системами и многими УС, поэтому конкретная конфигурация Системы может содержать много модулей СКМ и ИКМ. Эти модули разрабатываются под конкретное ПО УС и конкретные биллинговые системы.

Функции этих модулей кратко могут быть сведены к следующей технологии, которая схематически изображена на рис. 2.

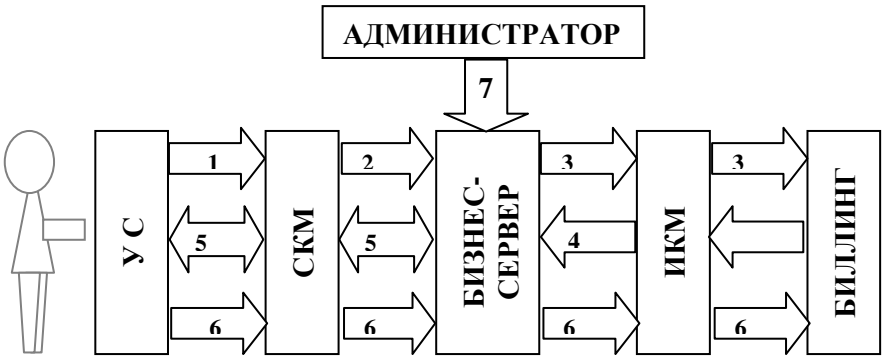


Рис.2. Схема выполнения платежей

1. В процессе функционирования Системы плательщик в диалоге с УС выбирает биллинговую систему, с которой ему необходимо связаться, и тем или иным способом вводит свой идентификатор (кредитную или другую карту, идентификационный номер, номер мобильного телефона и пр.). 2. СКМ передаёт введённую клиентом информацию на сервер. 3. ИКМ, постоянно запрашивая сервер Системы о поступивших запросах к биллингу, получает эту информацию и передаёт запрос о задолженностях клиента на «свой» биллинг. 4. Если у идентифицированного клиента есть задолженности в биллинге, то вся необходимая информация возвращается на сервер Системы, который на основании этой информации формирует сценарий дальнейшего диалога с клиентом. 5. Под управлением сервера, посредством СКМ ПО УС осуществляет этот сценарий взаимодействия с клиентом. 6. После выполнения заданного сценария и проведения всех технических действий, связанных с платёжной операцией по карте, УС передаёт информацию о проведённых операциях на сервер Системы, который в свою очередь транслирует её в биллинг. Вся специфика выполнения конкретных платежей отражается в сценарии, который формируется на сервере Системы. Настройку различных сценариев выполнения платежей осуществляет администратор Системы (7). Для этого в Системе разработан специальный язык настройки сценариев.

Язык Настройки Сценариев (ЯНС)

ЯНС — это процедурный язык, оперирующий такими понятиями, как тип экрана УС, поля и управляющие элементы (кнопки) экрана, пользовательские меню, переходы между экранами, подключённая биллинговая система и т.д. С помощью ЯНС можно описать сценарии для различных видов платежей в зависимости от технологии, принятой в конкретном биллинговом центре.

Описание сценария – скрипт считывается по мере необходимости (при этом проверяется его синтаксис и выдаются сообщения об ошибках) сервером Системы, преобразуется во внутреннее представление и интерпретируется в процессе обработки запросов устройств самообслуживания.

Центральным понятием языка является **сессия**. Под *сессией* будем понимать совокупность данных, связанных с одним устройством самообслуживания в течение одного сеанса обслуживания. Сессия создаётся в памяти сервера Системы в момент начала сеанса обслуживания и закрывается при окончании сеанса. В каждый данный момент времени работы сервера Системы может существовать несколько сессий при наличии нескольких одновременно обслуживаемых устройств самообслуживания. Каждой сессии при её открытии в Системе присваивается уникальный идентификатор.

Каждая сессия может находиться в одном из специфицированных в скрипте состояний. Такое состояние приблизительно соответствует некоторому виду экрана в устройстве самообслуживания. Поэтому в тексте скрипта для задания состояния используется ключевое слово **screen**.

Для каждого состояния сессии может быть описан набор переменных предопределённых типов и реакции на предусмотренные в Системе события. Переменные используются как хранилище данных о сессии. Реакции на события записываются в виде последовательности операторов С-подобного языка. В системе предусмотрен фиксированный набор событий, таких как нажатие на кнопки устройства самообслуживания, ошибочные ситуации, ожидание ответа биллинга, аварийное завершение сеанса со стороны УС.

ЯНС использует синтаксис и лексику, аналогичные языку программирования С. В языке используются три предопределённых ти-

па данных: **int**, **double**, **text**. Тип **int** предназначен для представления целых значений и использует 32-разрядные целые числа. Тип **double** предназначен для представления числовых значений, требующих в качестве внутреннего представления 64-разрядные вещественные числа. Тип **text** используется для представления текстовых данных длиной не более 1023 байт.

В качестве операторов языка могут использоваться такие конструкции, как **выражение** (присваивание, вызов функции); условный оператор **if**; циклы **for**, **while**; передача управления **break, continue, return**; переход в новое состояние **screen**; составной оператор { } с описанием локальных переменных предопределённых типов.

Для выполнения однотипных действий предусматривается использование функций (подпрограмм). Функции могут быть описаны как часть состояния, а также как общие для всех состояний. В функции, описанной в состоянии, доступны его параметры. Функция имеет тип значения и специфицированный состав параметров с типами. Параметры передаются по значению, т.е. не могут быть изменены вне функции. Тело функции задаётся как составной оператор.

Выражения строятся из констант предопределённых типов, переменных предопределённых типов, вызовов функций с фактическими параметрами, вызовов встроенных функций работы с различными объектами, и стандартных операций (сложение, вычитание, умножение, деление, текстовая конкатенация и пр.). В выражениях используются скобки для задания порядка выполнения, отличного от диктуемых приоритетами операций.

В процессе функционирования сервера Системы в памяти программы хранится текущая информация об активных сессиях и активных биллингах, а также некоторых других обрабатываемых данных. Эту информацию не следует путать с содержимым базы данных Системы. Содержимое базы данных хранится постоянно (в некотором смысле) (модифицируется при переходе сессии в определённые состояния), а текущая информация отражает состояние программы. Структура данных приведена на рис. 3.

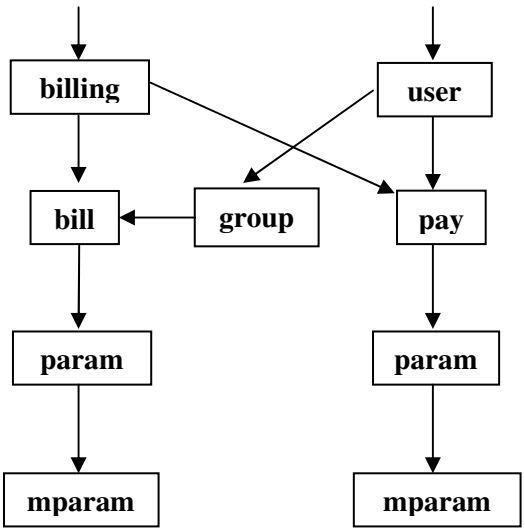


Рис. 3. Структура текущих данных

Наименования компонент на рис.3 соответствуют встроенным в ЯНС именам объектов:

- **user** — запись о сессии. Создаётся при начале работы клиента с устройством самообслуживания;
- **billing** — запись об активном клиенте биллинга. Создаётся при выполнении запроса клиента биллинга, при условии, что он ещё не зарегистрирован;
- **group** — группа однотипных платежей. Используется для группировки платежей по типу и для удобства размещения на экране устройства самообслуживания;
- **bill** — платёж, выставленный биллингом к оплате;
- **pay** — платёж, выбранный клиентом для оплаты;
- **param** — параметр платежа;
- **mparam** — вариант значения параметра платежа.

Стрелки на рисунке означают, что для каждого объекта, из которого выходит стрелка, может существовать список объектов (возможно пустой) типа, указанного стрелкой.

Таким образом, мы имеем в системе один список биллингов (**billing**), один список сессий (**user**), для каждой сессии – список выбранных платежей (**user => pay**), для каждого платежа – список параметров (**pay => param**) и т.д.

Описанная структура данных модифицируется в процессе работы сервера Системы как автоматически, в результате возникновения некоторых событий (создание сессии, подключение биллинга и т.д.), так и явно в результате выполнения программы скрипта. Для этого предусмотрен обширный набор встроенных функций, обеспечивающих создание, модификацию и удаление объектов описанной структуры.

Реализация проекта

Данный проект был реализован с помощью Генератора проектов в ЗАО «СмартКард-Сервис» (проект TellMe-MassPay), как расширение и дальнейшее развитие Единого гибридного программного обеспечения банкоматов (TellMe) разработки этой же компании.

СИСТЕМА МОБИЛЬНОГО БАНКОВСКОГО ОБСЛУЖИВАНИЯ

И.Л. Гринёв, А.А. Логинов, , А.Н. Широков, Н.И. Широков

Проблема быстрых безналичных расчётов и предоставления оперативного доступа к банковским счетам клиентов во всём мире приобретает всё большее значение по мере роста информационной инфраструктуры и требований клиентов.

Распространённым методом решения задачи обслуживания клиента on-line является применение банковских карт в качестве средства идентификации клиента, а иногда и в качестве носителя информации об остатке средств на карт-счёте. Наряду с этим традиционным методом в последнее время всё шире применяются мобильные телефоны как средство доступа к банковским счетам клиентов. Мобильный телефон является универсальным средством идентификации клиента, с помощью которого возможно реализовать весь набор услуг, предоставляемых держателям банковских карт, существенно его расширить и обеспечить надлежащий уровень защиты информационных потоков с целью практически полного исключения финансовых рисков участников расчётов. Высокая степень унификации стандартов связи сотовых сетей позволяет использовать для предоставления оперативного банковского сервиса аппарат мобильной связи вне зависимости от географического месторасположения клиента или банка. Применение мобильных телефонов на рынке банковских и расчётных услуг позволяет существенно упростить процедуру привлечения клиентов и снизить расходы на развитие сети обслуживания. Мобильный телефон может быть ассоциирован с банковским счётом. При этом стоимость технического и программного обеспечения для использования в качестве средства оплаты мобильных телефонов, как минимум не превышает стоимости обеспечения приёма платежей с банковских карт. Нами был разработан с помощью Генератора проектов работающий макет системы мобильного банковского обслуживания (далее СМБ – система мобильного банкинга) при участии компаний Intel, МТС, «ОРГА

Зеленоград», ЗАО «СмартКард-Сервис». Ниже приводится его описание.

Описание системы мобильного банкинга

СМБ представляет собой программно-аппаратный комплекс, построенный в архитектуре «многоуровневый клиент-сервер». Основной прикладной задачей, которую решает СМБ, является обеспечение с помощью мобильных телефонов on-line доступа клиентов к своим банковским счетам с целью выполнения безналичных банковских и торговых операций в режиме самообслуживания. В информационном плане задача мобильного банкинга сводится к организации защищённого обмена данными между клиентами и их банками. Технология обмена данными между клиентом и его банком определяется архитектурой системы (см. рис. 1) и реализуется на двух уровнях иерархии. На первом уровне клиенты с помощью своих мобильных телефонов взаимодействуют с бизнес-сервером СМБ, отправляя ему форматированные запросы и получая соответствующие ответы. На втором уровне запросы клиентов обрабатываются бизнес-сервером и формируются соответствующие ответы к отправке на мобильные телефоны клиентов.

При создании мультибанковской системы расчётов бизнес-сервер может устанавливаться и эксплуатироваться как в одном из банков-участников, так и в независимой процессинговой компании, являющейся в этом случае провайдером информационных услуг. Банки-участники регистрируются в центральной базе данных СМБ, при этом между бизнес-сервером и автоматизированными системами банков-участников устанавливаются программно-информационные интерфейсы. Эти интерфейсы необходимы в первую очередь для доступа к счетам клиентов банков-участников. Реализация таких интерфейсов может быть индивидуальна для каждого банка-участника. Возможен вариант ведения счетов клиентов банков непосредственно в центральной базе данных СМБ. В этом случае информационное взаимодействие бизнес-сервера СМБ и автоматизированной системы банка-участника может осуществляться в режиме off-line.

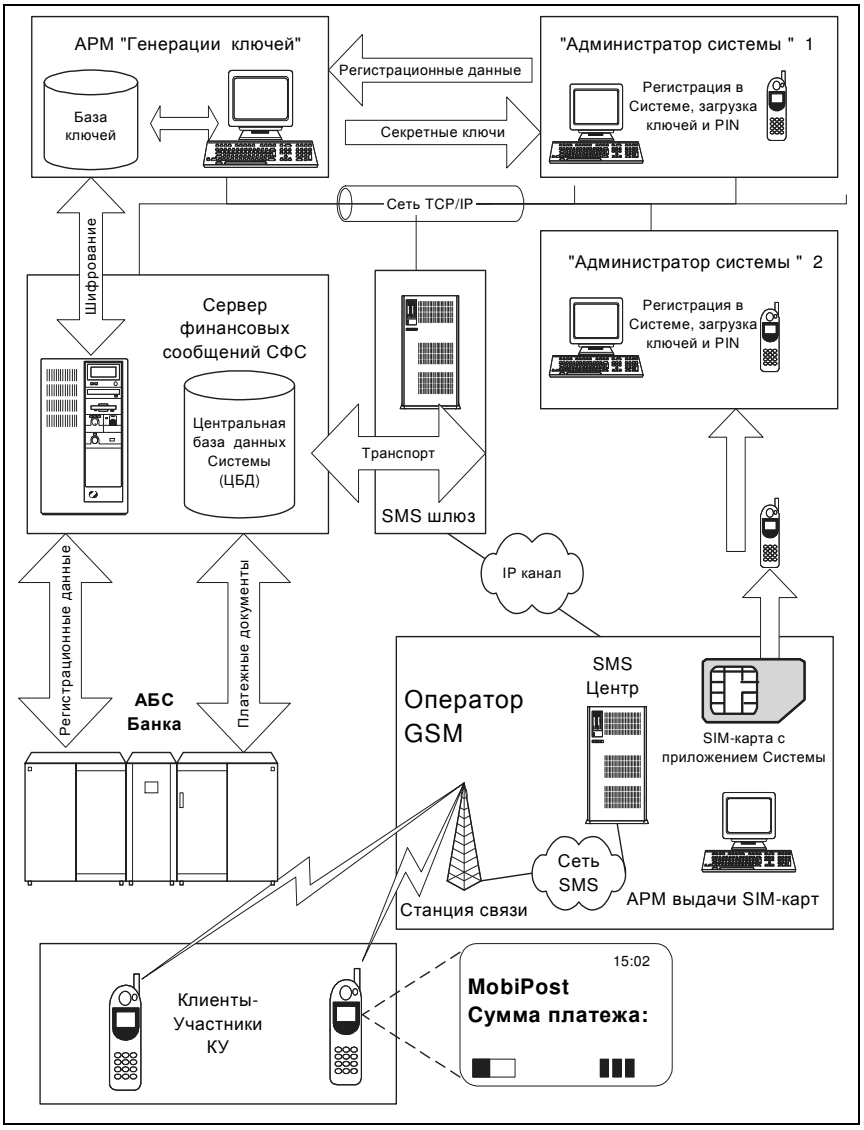


Рис. 1. Архитектура СМБ

Клиенты банков-участников СМБ также регистрируются в центральной базе данных, при этом с мобильным телефоном (SIM-модулем) клиента-участника ассоциируется его банковский счёт (счета). Кроме этого на SIM-модуль клиента-участника должно быть загружено специализированное программное обеспечение системы, а также данные, необходимые для обеспечения режима информационной безопасности.

Взаимодействие клиентов-участников с бизнес-сервером СМБ осуществляется путём обмена защищённой информацией в виде форматированных стандартных SMS-сообщений (**финансовых сообщений СМБ**). В общем случае финансовые сообщения, поступающие в бизнес-сервер СМБ, содержат инструкции клиентов-участников по управлению их счетами в банках-участниках. Бизнес-сервер обрабатывает входящие финансовые сообщения и транслирует их в автоматизированные системы банков-участников через соответствующие интерфейсы в режиме реального времени. На основании ответов автоматизированных систем банков-участников бизнес-сервер СМБ формирует исходящие финансовые сообщения и отправляет их на мобильные телефоны клиентов.

Задачи мобильной коммерции

Особое место в СМБ занимают операции безналичной оплаты товаров/услуг с помощью мобильных телефонов клиентов-участников. Эти операции представляют собой обмен финансовыми сообщениями между двумя клиентами-участниками (продавцом и покупателем) через бизнес-сервер СМБ. В общем случае средствами СМБ могут быть реализованы и более сложные технологические схемы, в которых участвуют более двух клиентов-участников. В настоящее время в СМБ реализованы платёжные операции трёх типов:

- клиент-участник (покупатель) инициирует безналичный платёж в адрес другого клиента-участника (продавца); транзакция выполняется без участия продавца товара/услуги;
- клиент-участник (продавец) инициирует платёжную транзакцию, выставляя средствами СМБ электронный счёт на оплату товара/услуги другому клиенту-участнику (покупателю); покупатель

подтверждает платёж, аутентифицирует себя вводом PIN, оба участника получают сообщения о выполнении транзакции;

- то же, что и второй вариант, но платёжная транзакция инициируется покупателем.

Для участия в платёжных транзакциях в качестве продавцов клиенты-участники СМБ могут использовать клиентские модули четырёх типов:

- Интернет-магазин;
- торговая точка;
- банкомат (информационный киоск);
- мобильный телефон клиента-участника.

Информационная безопасность

Уровень защиты информационных потоков в СМБ зависит от потребностей заказчика и может включать в себя:

- шифрование финансовых сообщений;
- снабжение финансовых сообщений электронными цифровыми подписями (ЭЦП);
- процедуры взаимной аутентификации клиентов-участников и сервера СМБ.

Для реализации данной функциональности может быть использован широкий спектр стандартных методов и алгоритмов.

Основные функции СМБ

1. Регистрация клиентов-участников и банков-участников СМБ в центральной базе данных.
2. Защищённый доступ клиентов-участников к их банковским счетам с возможностью получения актуальной информации о состоянии этих счетов и проведения банковских операций в режиме реального времени.
3. Проведение финансовой транзакции безналичной оплаты с участием двух клиентов-участников в режиме реального времени.
4. Проведение защищённой финансовой транзакции выдачи наличных средств на банкомате (аналогично торговой точке). Программное обеспечение банкомата позволяет одновременно про-

водить транзакции как с мобильными телефонами, так и с банковскими картами (магнитными и микропроцессорными).

5. Ведение базы данных клиентов-участников, банков-участников, условий обслуживания (тарифы и индивидуальные уровни комиссионных отчислений, периоды обслуживания, «чёрные» списки и т.д.)
6. Ведение полных журналов событий СМБ и построение финансовой, бухгалтерской и статистической отчётности.
7. Информационное взаимодействие как в режиме on-line, так и пакетном режиме (off-line) с внешними системами (банковские автоматизированные системы, системы учёта деятельности продавцов – склад, бухгалтерия, касса, биллинговые системы операторов мобильной связи и т.д.).

Реализация макета СМБ

Макет системы был реализован средствами Генератора проектов. Объём исходного текста описания проекта составил порядка 110 Kb. Общий объём программного обеспечения СМБ в результате генерации системы 3 MB. Конфигурация оборудования и программных компонент СМБ определяется заказчиком в процессе её внедрения и дальнейшей эксплуатации. Ниже приведены общие данные, которые могут быть изменены в зависимости от конкретных условий.

Программное обеспечение сервера. Сервер приложения СМБ, функционирующий под управлением ОС Windows NT/2000, UNIX, Open VMS.

Информационное обеспечение. Центральная база данных СМБ, функционирующая под управлением СУБД MS SQL Server, Oracle, Sybase, Inormix, RDB и т.д.

Оборудование сервера. Состав оборудования определяется потребной производительностью СМБ и зависит от объёма передаваемой информации в часы пиковой нагрузки. Рекомендуется реализовывать взаимодействие бизнес-сервера СМБ с SMS-центром по IP-каналу, однако для этих целей можно применять и GSM-модемы.

Для торговой точки возможны четыре варианта конфигурации:

- модуль продавца функционирует на персональном компьютере, (возможно, в составе контрольно-кассовой машины) под управлением Windows 95, 98, NT и выше. Подключён по IP-Сети к бизнес-серверу СМБ;
- модуль продавца функционирует на персональном компьютере , (возможно, в составе контрольно-кассовой машины) под управлением Windows 95, 98, NT и выше. Оснащён мобильным телефоном или GSM-модемом, взаимодействует с бизнес-сервером через GSM-сеть;
- GSM-POS (например, типа NURIT), со встроенными GSM-модемом и чековым принтером;
- мобильный телефон клиента-участника.

Для банкомата программное обеспечение и оборудование управляющего компьютера конфигурируется аналогично торговой точке. В зависимости от конфигурации возможно подключение модулей для обслуживания банковских карт международных платёжных систем (VISA, Europay) и локальных платёжных систем на основе технологии DUET. Возможна также разработка специализированных модулей для приёма других платёжных средств, специфицированных заказчиком.

Мобильный телефон клиента-участника. GSM-телефон (фаза 2+), SIM-модуль ORGA SIMtelligenceR Java-SIM 72/32 с загруженным программным обеспечением Системы.

СИСТЕМА ИНТЕРНЕТ – ПЛАТЕЖЕЙ

И.Л. Гринёв, А.А. Логинов, Н.И. Широков

Введение

Коммерческое использование Интернет приобретает всё более грандиозные масштабы. Уже сегодня большая часть коммерческих сделок заключается с использованием тех или иных Интернет-технологий, начиная от электронной почты и заканчивая электронными платежами.

В соответствии с этим стремительно развиваются и программно-технические средства обслуживания коммерческой деятельности в Интернет, в частности средства финансового обслуживания коммерческих сделок.

Так, например, совершение покупки в Интернет-магазине подразумевает сегодня не только автоматическую генерацию счёта, резервирование товара, но зачастую и непосредственную «on-line» оплату покупки.

Несмотря на обилие внешне различных механизмов такой оплаты, формально все они сводятся к формированию поручения на перечисление денежных средств с одного банковского счёта (счёта Клиента) на другой (счёт Продавца).

Схематически механизм оплаты можно представить следующим образом:

Рис.1. Схема взаимодействия

Ключевым моментом реализации такого механизма является обеспечение безопасного доступа Клиента к его банковскому счёту. Стандартом в данном вопросе считается использование некоторой так называемой платёжной системы, основанной на использовании банковских карт: микропроцессорных или карт с магнитной полосой.

Описанная в данном документе Система Интернет-Платежей (СИП) представляет собой проект комплекса программно-аппаратных средств, позволяющих производить «on-line» оплату покупок (услуг) в Интернете, используя микропроцессорные карты системы DUET. Данный проект был разработан при непосредственном участии ЗАО «СмартКард-Сервис» с помощью инструментального комплекса Генератора проектов.

Технология расчётов

Основной функцией системы является оплата покупок (услуг) в Интернет-магазинах с помощью микропроцессорных карт AC DUET через сеть Интернет. При этом средства переводятся с клиентской карты Клиента на торговую карту Продавца, которая физически находится у Оператора расчётов. Ниже приводится детализация технологии выполнения таких расчётов.

Технологической основой системы является технология расчётов с использованием микропроцессорных карт системы DUET. Платёжная транзакция (расчёт между Продавцом и Покупателем) по данной технологии производится во взаимодействии двух микропроцессорных карт Карты Клиента (Клиента) и Торговой Карты (карты Продавца) и не требует связи с сервером системы DUET. Загрузка средств на Карту Клиента с расчётного счёта в банке производятся по данной технологии во взаимодействии карты с сервером системы DUET. Инкассация средств с Торговой Карты на расчётный счёт в банке производятся по данной технологии во взаимодействии карты с сервером системы DUET.

Расчёты в СИП производятся через сеть Интернет во взаимодействии следующих программно-технических подсистем:

- подсистемы Оператора расчётов;
- Интернет-магазина;

- модуля Клиента.

Кроме того, для осуществления загрузки средств, инкассации и других операций с банковским счётом подсистема Оператора расчётов взаимодействует с автоматизированной системой банка (сервером АС DUET) и должна иметь для этого специальный канал связи.

Рис. 2. Схема взаимодействия модулей.

Полный технологический цикл оплаты покупок в Интернет-магазинах предполагает взаимодействие всех подсистем СИП. На первой фазе этой операции с помощью модуля Клиента и во взаимодействии с Интернет-магазином формируется виртуальная корзина покупок, и Интернет-магазин выставляет Покупателю счёт на оплату. Вторая фаза операции – собственно оплата корзины, в процессе которой средства переводятся с карты Клиента на торговую карту Продавца. Третья фаза – подтверждение оплаты покупки, осуществляемое во взаимодействии Интернет-магазина и подсистемы Оператора расчётов.

Архитектура системы

Базовой архитектурой СИП является архитектура «*трёхуровневый клиент-сервер*». Для этой архитектуры характерно наличие трёх основных взаимосвязанных компонент:

- **централизованной базы данных** под управлением реляционной СУБД;

- **программного сервера приложений (бизнес-сервера),** выполняющего запросы к базе данных через стандартный программный интерфейс, поддерживаемый конкретной СУБД, реализующий язык SQL;
- **клиентских модулей** – программ, которые, обращаясь к серверу приложений, инициируют выполнение содержательных запросов к базе данных и визуализируют результаты их исполнения на экранах удалённых рабочих мест пользователей системы в режиме on-line.

Подсистема Оператора расчётов и Интернет-магазин в СИП в общем построены в соответствии с этой архитектурой. Но кроме общих (перечисленных выше) компонент в их состав также входят:

- программный сервер безопасности;
- Web-сервер.

Программные серверы безопасности обеспечивают взаимную аутентификацию пользователей и компонент системы, а также шифрование информационных обменов между клиентскими модулями и серверами приложений.

Наличие Web-сервера в общей архитектуре системы позволяет использовать специфические реализации клиентских модулей на базе стандартных браузеров.

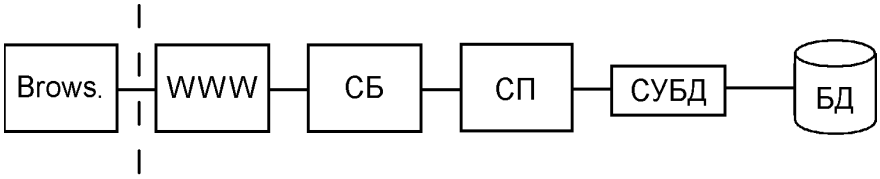


Рис. 3 Место сервера безопасности и WEB-сервера в архитектуре СИП

В состав подсистемы Оператора расчётов входит ещё одна специализированная компонента – **сервер торговых карт (СТК)**.

Именно эта компонента обеспечивает интеграцию СИП с платёжной системой AC DUET.

Дальнейшее развитие СИП предполагает подключение и других платёжных систем (в т.ч. использующих магнитные карты), для чего в системе будут использоваться другие программные компо-

ненты, которые в общей архитектуре системы займут то же место, что и сервер торговых карт.

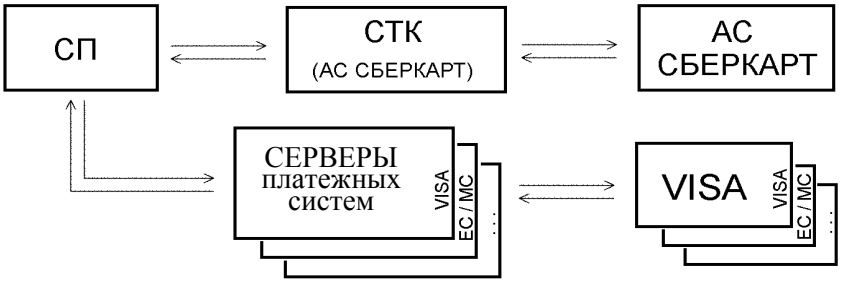


Рис. 4. Схема взаимодействия сервера приложений с карточными платёжными системами

Необходимо отметить, что в процессе функционирования СИП Интернет-магазин может выполнять запросы к подсистеме Оператора расчётов (например, запрос подтверждения оплаты покупки), т.е. можно говорить о том, что сервер приложений Интернет-магазина является клиентом сервера приложений подсистемы Оператора расчётов (и т.п.).

Таким образом, архитектуру СИП в целом можно классифицировать как «*многоуровневый клиент-сервер*».

Программное обеспечение компонент СИП

Программный комплекс СИП включает в себя три самостоятельные подсистемы, взаимодействующие между собой. Это:

- оператор расчётов,
- Интернет - магазин (макет),
- клиентский модуль.

Подсистема **Оператор расчётов** включает в себя:

- бизнес-сервер (БС);
- сервер безопасности (СБ);
- сервер торговых карт (СТК);
- Web-сервер (WWW);
- модуль администратора СИП (Админ.).

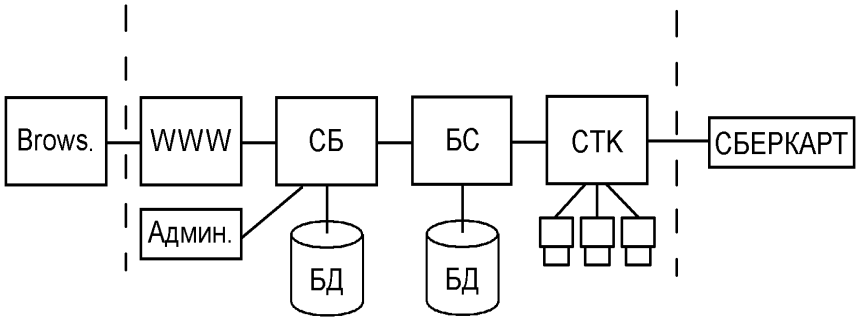


Рис. 5. Схема взаимодействия модулей подсистемы оператора расчётов

Бизнес-сервер (БС) подсистемы Оператора расчётов предназначен для фиксирования (в централизованной базе данных СИП) информации обо всех проведённых операциях, ведения справочников НСИ и учёта (на специализированных счетах в базе данных) взаимных финансовых отношений участников СИП.

Также бизнес-сервер (используя базу данных СИП) обрабатывает информационные запросы клиентских модулей и Интернет-магазинов.

Сервер безопасности (СБ) подсистемы Оператора расчётов обеспечивает взаимную аутентификацию подсистемы и её пользователей, а также шифрование/дешифрование информационных обменов между сервером приложений и клиентскими модулями.

На сервере безопасности регистрируются пользователи подсистемы. При регистрации пользователям присваиваются необходимые права и полномочия, для каждого из них генерируется ключевая информация, которая записывается на индивидуальную дискету (либо другой носитель). Дискета используется для работы с модулем администратора СИП.

Сервер торговых карт (СТК) – специализированная компонента подсистемы Оператора расчётов, предназначенная для выполнения операций с микропроцессорными картами (торговыми картами AC DUET).

Сервер торговых карт в процессе оплаты покупки обеспечивает взаимодействие клиентской карты с требуемой торговой картой,

выполняет (во взаимодействии с сервером АС DUET) инкассацию торговых карт, участвует в выполнении сервисных операций с картой Клиента, выполняемых на модуле Клиента.

Web-сервер (WWW) подсистемы Оператора расчётов предназначен для взаимодействия с клиентскими модулями, реализованными на базе стандартных браузеров. В текущей версии СИП Web-сервер не используется.

Модуль администратора СИП (Админ.) – представляет собой программу, взаимодействующую с бизнес-сервером подсистемы (через сервер безопасности) по внутренним содержательным протоколам СИП, как в локальном, так и в удалённом режимах.

Реализация **Интернет-магазина** предназначена для демонстрации Интернет-платежей в рамках макета СИП. В общем случае в СИП могут быть интегрированы Интернет-магазины других разработчиков.

Программный комплекс Интернет-магазина включает в себя:

- бизнес-сервер (БС);
- сервер безопасности (СБ);
- Web-сервер (WWW);
- модуль администратора Интернет-магазина (Админ.).

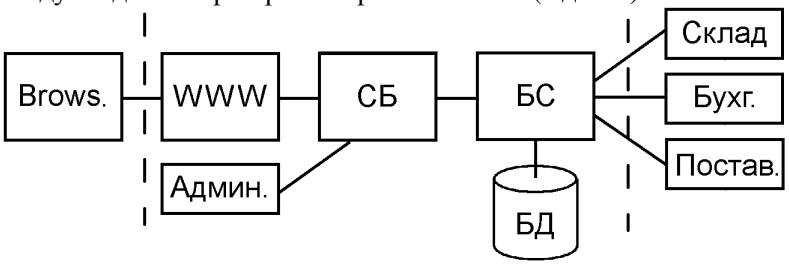


Рис. 6. Схема взаимодействия модулей Интернет-магазина

Бизнес-сервер (БС) данной подсистемы предназначен для выполнения содержательных запросов к централизованной базе данных Интернет-магазина. (Наиболее важными информационными объектами, хранящимися в базе данных Интернет-магазина, являются каталог товаров и виртуальные покупательские корзины.)

Сервер безопасности (СБ) Интернет-магазина по устройству и функциям аналогичен Серверу безопасности подсистемы Оператора расчётов.

Web-сервер (WWW) предназначен для взаимодействия с модулями Клиента, реализованными на базе стандартного браузера.

Модуль администратора Интернет-магазина (Админ.) – представляет собой программу, взаимодействующую (через сервер безопасности) с бизнес-сервером подсистемы по внутренним содержательным протоколам СИП как в локальном, так и в удалённом режимах.

Модуль Клиента является клиентским модулем, реализованным на базе стандартного браузера.

СИСТЕМА УРЕГУЛИРОВАНИЯ ЗАДОЛЖЕННОСТЕЙ

Л.Л. Вышинский, И.Л. Гринёв, Н.И. Широков, В.И.Щедров

Взаимные задолженности субъектов экономической деятельности (СЭД) и неплатежи по этим задолженностям были и попрежнему остаются весьма болезненной проблемой для нашей экономики. Причиной возникновения неплатежей является недостаток оборотных средств на счетах предприятий и организаций. Одним из решений этой проблемы является проведение взаимных зачётов по долгам предприятий друг другу. Однако взаимные зачёты возможны не только между парами, но между тремя и более субъектами по «круговым» (циклическим) долгам (А должен В, В должен С, С должен А). Для реализации подобного решения проблемы неплатежей в 1999 г. по инициативе Российского Промышленного Банка нами была разработана автоматизированная система «РПБ клиринг».

Назначением «РПБ клиринга» была подготовка и организация взаимных расчётов по урегулированию задолженностей. Основные задачи, которые ставились перед системой, это: сбор информации о взаимных задолженностях СЭД, выявление возможностей сбалансированного полного или частичного погашения взаимных задолженностей, подготовка необходимых документов проведения взаимных расчётов. Собственно проведение взаимных расчётов должно осуществляться с помощью уполномоченных на то банковских или других финансовых учреждений. Организация взаимных расчётов с помощью системы «РПБ клиринг» может осуществляться на региональном и на межрегиональном уровне.

Организация и подготовка урегулирования между СЭД в системе «РПБ клиринг» осуществляется на основании заявок от организаций о просроченных задолженностях.

Приём и обработка заявок от участников урегулирования проводится в рамках отдельных сессий в сроки, согласованные с заинтересованными органами администрации регионов. Сессии урегули-

рования взаимной просроченной задолженности системе «РПБ клиринг» нумеруются в рамках региона.

Порядок подготовки урегулирования взаимных задолженностей с помощью системы «РПБ клиринг» предусматривает для каждой очередной сессии следующие этапы:

- приём заявок от участников;
- урегулирование разногласий по заявкам между участниками;
- вычисление возможных вариантов урегулирования задолженностей;
- согласование сумм урегулирования;
- предоставление результатов сессии участникам урегулирования.

Порядок приёма и обработки заявок предусматривает объявление следующих дат для каждого этапа очередной сессии, которые определяются внесистемным (административным) способом и используются при организации работы отдельных сессий:

- даты начала приёма заявок;
- даты окончания приёма заявок;
- срока урегулирования разногласий по заявкам;
- сроков проведения вычислений и согласования сумм взаимных расчётов;
- даты предоставления окончательных результатов сессии участникам урегулирования.

Заявка о просроченной дебиторской и кредиторской задолженности подаётся от имени заявителя, имеющего эти задолженности. Заявка должна содержать полный набор реквизитов заявителя и списки просроченных дебиторских и/или кредиторских задолженностей по одному или нескольким контрактам (договорам).

Заявка о просроченной дебиторской и кредиторской задолженности должна содержать следующий набор реквизитов заявителя:

- ИНН заявителя;
- полное наименование заявителя;
- код региона заявителя;
- дата предоставления заявки;
- банковские и другие реквизиты заявителя.

Для каждой отдельной суммы просроченной дебиторской или кредиторской задолженности в заявке должны быть указаны следующие данные:

- ИНН дебитора (кредитора);
- полное наименование дебитора (кредитора);
- сумма задолженности;
- номер контракта (договора), по которому возникла задолженность;
- дата контракта.

Ввод заявок о дебиторской или кредиторской задолженности может осуществляться «вручную» на основании поданных бумажных форм заявок или из файлов, предоставленных заявителями. Файлы с заявками от организаций формируются с помощью специализированной автономной программы, распространяемой операторами урегулирования.

Для регистрации участников в системе «РПБ клиринг» ведётся специальный «Справочник участников урегулирования». Основанием для включения предприятия или организации в этот справочник является заявка организации о просроченной кредиторской и дебиторской задолженности. «Справочник участников урегулирования» ведётся в электронном виде в базе данных системы. При необходимости он может быть распечатан на бумажном носителе. Для каждого участника в справочнике фиксируются реквизиты, перечисленные выше. Кроме того, для каждого участника в справочнике ведётся дата последней модификации его реквизитов.

При вводе заявки по значению ИНН заявителя (уникальный ключевой реквизит) система автоматически проверяет наличие аналогичной записи в «Справочнике участников урегулирования». Если такой записи нет, то заявитель регистрируется в справочнике. Если в справочнике уже зарегистрирован участник с таким же ИНН и дата вводимой заявки превышает дату последней модификации для данного участника, то новая заявка рассматривается как поручение на модификацию всех указанных в ней реквизитов заявителя.

При вводе заявки по каждой сумме просроченной дебиторской (кредиторской) задолженности система автоматически проверяет по ИНН дебитора (кредитора) наличие регистрационной записи в «Справочнике участников урегулирования». Если такой записи нет,

то информация о соответствующем СЭД включается в «Справочник участников урегулирования». При этом для него фиксируются лишь ИНН и наименование, взятые из заявки. Если в «Справочнике участников урегулирования » уже зарегистрирован участник с таким же ИНН, а наименование в заявке отличается от записи в справочнике, то это не даёт основания для модификации справочника.

Для регистрации каждой заявленной просроченной задолженности в системе ведётся «Журнал заявленных просроченных задолженностей». Журнал ведётся в рамках каждой сессии урегулирования взаимной задолженности СЭД. «Журнал заявленных просроченных задолженностей» ведётся в электронном виде в базе данных системы. При необходимости журнал задолженностей может быть распечатан на бумажном носителе.

При вводе заявки о просроченной дебиторской и кредиторской задолженности все заявленные в ней суммы регистрируется в «Журнале заявленных просроченных задолженностей» каждая отдельной строкой. При этом каждой задолженности автоматически присваивается свой уникальный номер.

В «Журнал заявленных просроченных задолженностей» системы «РПБ клиринг» для каждой задолженности вносятся следующие реквизиты:

- порядковый номер (из заявки);
- уникальный присвоенный номер заявленной задолженности;
- ИНН заявителя;
- ИНН дебитора;
- полное наименование дебитора (из заявки);
- ИНН кредитора;
- полное наименование кредитора (из заявки);
- сумма задолженности;
- номер контракта (договора), по которому возникла задолженность;
- дата контракта;
- дата предоставления заявки.

Зарегистрированная задолженность может быть использована во взаимных расчётах. В системе существует несколько режимов расчёта. Первый способ предъявляет достаточно жёсткие условия к

контрагентам. Этот способ расчёта включает заявку в расчёт только тогда, когда она подтверждена полностью контрагентом. При регистрации новой задолженности автоматически проверяется: существует ли среди ранее зарегистрированных задолженностей такая, которая подтверждает вновь регистрируемую. Если подтверждающая задолженность найдена, то и вновь регистрируемая задолженность и подтверждающая получают статус «подтверждённые». Взаимно подтверждающими считаются такие задолженности, у которых совпадают все реквизиты: ИНН и полное наименование контрагентов, номер контракта (договора), дата контракта и сумма задолженности.

При втором режиме расчёта участвуют полностью подтверждённые заявки и заявки, у которых совпадают только ИНН и полные наименования контрагентов. При этом номер контракта, дата и сумма могут отличаться. В этом режиме заявка считается подтверждённой частично и на сумму, составляющую минимум из двух указанных сумм.

Наконец, третий режим расчёта вообще не требует подтверждения заявки контрагентом.

По истечении срока приёма заявок каждому заявителю может быть отправлено извещение о приёме заявки. В извещении должны указываться:

- ИНН заявителя;
- общая сумма заявленной дебиторской задолженности;
- общая сумма заявленной кредиторской задолженности;
- сумма подтверждённой дебиторской задолженности;
- сумма подтверждённой кредиторской задолженности;
- сумма дебиторских задолженностей, по которым нет подтверждения;
- сумма кредиторских задолженностей, по которым нет подтверждения;
- сумма дебиторских задолженностей, не согласующихся с данными контрагента;
- сумма кредиторских задолженностей, не согласующихся с данными контрагента;

- расшифровка всех сумм, по которым возникли разногласия с контрагентом.

По специальным запросам заявителям, которые не подтвердили какие-либо из заявленных другими участниками задолженностей, может направляться список этих задолженностей.

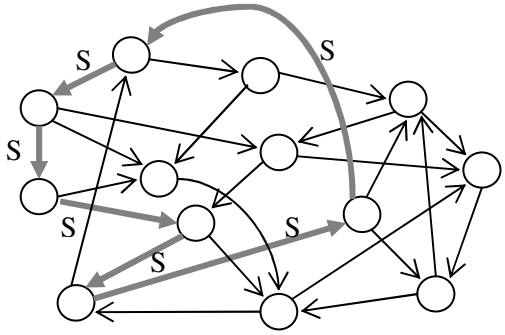
В сроки, отведённые правилами, заявители должны согласовать возникшие разногласия и представить необходимые модификации заявок по задолженностям для введения их в систему «РПБ клиринг».

После истечения срока, отведённого на урегулирование разногласий и разночтений, по заявленным задолженностям производятся вычисления сумм для урегулирования взаимных задолженностей. В вычислениях могут приниматься во внимание только взаимно подтверждённые заявленные задолженности.

Вычисление сумм урегулирования взаимных задолженностей сводится к построению так называемых «цепочек». Под «цепочкой» в системе «РПБ клиринг» понимается множество взаимных платежей между некоторой группой участников системы взаимного урегулирования, которая удовлетворяет условию равенства суммы дебетовых платежей «цепочки» сумме кредитовых платежей для каждого члена выделенной группы участников (сумма выплат равна сумме поступлений).

В построенных «цепочках» общие суммы взаимных платежей складываются из сумм просроченных задолженностей, которые упорядочены по дате контракта. Допускается частичное урегулирование отдельных задолженностей.

Построения «цепочек» сводится к задаче поиска «взвешенных» циклов на ориентированном графе с весами. Вес дуги графа $S_{i,j}$ отражает сумму задолженности i-го участника j-му. На рисунке приведён пример графа задолженностей участников расчётов.



Выделенный цикл отражает одну из возможных «цепочек». Для этого цикла должно удовлетворяться условие $S \le S_{i,j}$ для всех дуг, входящих в этот цикл. При построении «цепочек» на графе можно ставить различные оптимизационные задачи, например, задачу поиска цикла максимальной длины с ограничениями на весовые коэффициенты или задачу максимизации общей суммы урегулирования, или задачу максимальной компенсации задолженности государственным структурам. В системе «РПБ клиринг» применяются специально разработанные эвристические алгоритмы для решения подобных задач.

Вычисления «цепочек» могут осуществляться в несколько этапов, охватывая различные группы участников. Например, сначала могут быть проведены расчёты только по региональным расчётам, а после этого к оставшимся региональным задолженностям добавляются заявленные и подтверждённые межрегиональные задолженности и проводится расчёт новых «межрегиональных цепочек».

Для более эффективного устранения взаимных задолженностей в системе «РПБ клиринг» допускается создание «искусственных» задолженностей, которые бы «замыкали» недостающие звенья «цепочек» и которые могут рассматриваться как инвестиции в систему взаимного урегулирования. При этом допускается введение участников, которые не являются заявителями о дебетовых или кредито-

вых задолженностях, а их участие состоит в посредничестве между участниками взаимного урегулирования.

После получения приемлемых результатов расчётов по каждой из «цепочек» составляются документы урегулирования, определяющие суммы взаимных расчётов между участниками данной цепочки.

Если какие–либо заявленные задолженности не могут быть использованы в урегулировании или не попадают в реестр, то информация о них может быть направлена в федеральные или в другие региональные системы взаимных расчётов.

По результатам сессии составляется отчёт о найденных «цепочках» и урегулированных задолженностях. По окончании сессии «Журнал заявленных просроченных задолженностей», результаты расчётов («цепочки», документы урегулирования и пр.) перемещается в архив системы.

Система «РПБ клиринг» построена в рамках архитектуры «трёхуровневый клиент-сервер». Состав программных компонент системы включает:

- модуль администратора системы;
- модуль оператора взаиморасчётов;
- прикладной бизнес - сервер системы;
- базу данных системы.

Эти программные компоненты могут функционировать как на одном компьютере – сервере системы, так и на разных, объединённых в сеть (локальную или глобальную).

Важной технологической компонентой системы является автономная программа для формирования заявок на урегулирование задолженностей. Эта программа распространяется среди организаций-участников системы совместно с электронным «Справочником участников урегулирования». Такая технология позволила получать от организаций-участников заявки на урегулирование в электронном виде (на магнитных носителях или по электронной почте) и в некотором стандартном формате. Эти заявки загружаются в систему автоматически.

Клиентские модули (модуль администратора и модуль оператора) взаимодействует с бизнес-сервером системы посредством специализированного протокола. Этот протокол оперирует содержательными запросами на выполнение конкретных действий в рамках полномочий пользователя. Программный сервер системы может обеспечить выполнение самых сложных проверок полномочий пользователей на основе информации в базе данных и характера запроса. В свою очередь, сервер системы взаимодействует с сервером базы данных на уровне языка SQL.

Разработка «РПБ клиринг» велась средствами Генератора проектов, благодаря чему удалось небольшому коллективу быстро и качественно создать необходимое программное обеспечение этого проекта. Система прошла опытную эксплуатацию в республике Марий Эл, в эксперименте участвовали не только республиканские предприятия, но и ряд субъектов экономической деятельности из соседних областей России. Анализ результатов опытной эксплуатации системы "РПБ клиринг" продемонстрировал эффективность предлагаемой методики.

КОМПЛЕКСНАЯ АВТОМАТИЗАЦИЯ ПОСТАВКИ ТОВАРОВ

В. И. Шиленко

Одной из актуальных задач финансовых информационных технологий является автоматизация организаций, занимающихся выполнением заказов на поставку товаров. В зависимости от объёма номенклатуры товаров и объективно сложившейся технологии работы такого рода автоматизация может быть как сравнительно несложной задачей, так и длительным, наполненным подлинного драматизма процессом. В сложных случаях неизбежна индивидуальная разработка программного обеспечения, с наибольшей эффективностью отражающего специфику конкретной организации. Ниже приводятся методические аспекты автоматизации, которые могут быть полезными в аналогичных разработках.

Описание модели

В качестве базовой модели рассмотрим организацию, поставляющую запчасти к автомобилям по заказам юридических и физических лиц. В упрощённом виде бизнес-процесс состоит из следующих ниже этапов.

- Заказчик определяет перечень необходимых товаров с необходимыми реквизитами и отправляет заявку специалистам организации.
- Специалисты фирмы проверяют заявку, уточняют реквизиты товаров и оформляют заказ.
- Для заказчиков, с которыми происходит расчёт по каждому заказу, выставляется счёт и ожидается поступление средств.
- Для заказчиков, от которых поступила оплата или остаток средств достаточен для выполнения заказа (с возможным учётом овердрафта), отправляют заказ на исполнение; возможно объединение заказов одному поставщику в сводный заказ.

- Между поставщиком и специалистами фирмы по ряду позиций происходит согласование возможных изменений (цена поставки, замена номера, срок поставки, количество).
- По мере закупки исполнителем заказа товара у других поставщиков (поставщиков 2-го уровня) формируется груз для отправки на фирму; в качестве сопроводительных документов к нему прилагаются накладные; одновременно сведения об отправке груза передаются на фирму (груз при этом “в пути”).
- После прибытия груза на оперативный склад фирмы происходит проверка соответствия пришедших товаров грузоприёмочной ведомости; по каждой позиции происходит анализ соответствия цен поставки с планировавшимися закупочными ценами и “неубыточность” отпускных цен по отношению к закупочным.
- Для проверенных грузов строятся фактические накладные, счета-фактуры и товарные накладные для каждого заказчика; Сумма фактической накладной вносится на счёт заказчика в валюте баланса; там же учитываются транспортные расходы на доставку заказанных товаров заказчику, полученную от него оплату и т.д.
- При выдаче или отправке товаров заказчику все позиции накладной получают признак “Выдано”.

Реальная практика вносит в эту схему множество нюансов, которые должны отрабатываться в рамках автоматизированной системы организации.

Заказы на товары

Основные реквизиты заказа – номер (присваивается по нарастающей последовательности), дата, заказчик (ссылка на таблицу “Заказчики”), очередной номер заказа выбранного заказчика, курс рубля к доллару и евро, тип оплаты (наличная или безналичная оплата). Реквизиты товаров данного заказа – направление заказа, ссылка на каталог, номер по каталогу, наименование, количество, закупочная и отпускная цена с указанием валют, цена товара в рублях (для безналичных расчётов), признак состояния (стадия жизненного цикла товара), поля запросов/ответов для согласований с заказчиком и поставщиком.

При оформлении заказа – центрального объекта автоматизации – необходимо учитывать следующие обстоятельства.

- Необходимо чётко разграничивать первоначальное создание заказа и возможные в дальнейшем неоднократные его изменения; в целях безопасности и возможного аудита изменения реквизитов протоколируются.
- Курсы рубля к доллару и евро устанавливаются по текущим значениям в организации в зависимости от типа оплаты (безналичная или наличная).
- Тип оплаты устанавливается в зависимости от реквизитов заказчика.
- Поставщики (направления) заказа для каждого товара выбираются специалистом из динамически поддерживаемого списка.
- Каталоги товаров связаны с поставщиками.
- Номер товара выбирается из списка ранее заказывавшихся по данному каталогу (номера иногда содержат до 20 символов!) или вводится специалистом.
- Наименование товара выбирается из фиксированного перечня, что необходимо для перевода на английский или немецкий язык при формировании сводного заказа.
- Ввод или изменение закупочной цены приводит к вычислению отпускной цены в валюте каталога и в рублях (только при создании заказа!) с учётом коэффициента затрат для данного заказчика по данному направлению и скидок по выбранному каталогу.
- Признак состояния товара при создании заказа получает значение “Оформлено”; в дальнейшем он автоматически переходит в признаки “Сводный”; “Послано”; “Куплено”; “Не проверено”; “Проверено”; “Выдано”. С учётом жизненных коллизий возможны и признаки “Задержка заказчика”, “Отказ заказчика”, “Задержка поставщика”, “Отказ поставщика”. Изменение признака специалистом допустимо в строго определённых рамках и протоколируется.

Технология ввода данных заказа должны быть максимально автоматизирована, она должна блокировать ошибки пользователей и обеспечивать безопасность данных.

Сводные заказы

Сводные заказы формируются системой из всех товаров выбранного направления с признаком “Оформлено”. Как правило, это происходит с определённой периодичностью, но могут использоваться и дополнительные критерии (например, превышение определённой суммы). Поставщик, обслуживающий направление, по сути дела является диспетчером, получающим сводный заказ от фирмы и закупающим товары на специализированных складах в стране пребывания. Из-за сравнительно низкого уровня автоматизации в этих организациях работа протекает практически на бумажном документообороте. Это вынуждает применять специальные процедуры распределения закупленного товара по заказчикам. При построении сводных заказов целесообразно:

- строить заголовок с указанием всех каталогов данного заказа и количеством позиций по каждому из них; все позиции сводного заказа последовательно нумеровать;
- разбивать сводный заказ по каталогам и позиции внутри каждого из них сортировать по номеру товара; каждую такую часть снабжать уточняющими данными;
- хранить данные по наименованию отчёта (сводные заказы имеют различное оформление) и размещению отправляемого файла в таблице с описанием направлений.

Поставки

Обработка поставки – существенная фаза технологического процесса, эффективность которой поддерживается следующими решениями:

- поставляемые товары подразделяются на грузы, а грузы – на поставки по отдельным каталогам;
- к реквизитам груза относятся ожидаемая дата прибытия, направление, страна поставки, номер авианакладной, вес, статус, расходы на доставку груза от поставщика до фирмы, включая таможенные платежи; после прибытия и изменения статуса дата груза корректируется; для груза строится грузоприёмочная ве-

домость, транспортная накладная, таможенные спецификации различных видов;

- поставки по каталогам включают в себя номер товара, ссылку на заказчика, цену поставки, скидку, количество;
- при отсутствии ссылки на заказчика происходит поиск по хронологии заказов и количеству; далее каждая позиция поставки сопоставляется с позицией исходного заказа, причём признак товара переходит в состояние “куплено”;
- при неполном исполнении заказа в исходном заказе производится расщепление записи;
- для всех товаров груза проводится анализ соответствия цен поставки с закупочными ценами, в случае расхождения специалисты производят корректировку исходного заказа; анализируется рентабельность отпускных цен;
- товары, которые не были заказаны, но попали в поставочные документы (присланы по ошибке), выделяются в раздел “проблемные товары” грузоприёмочной ведомости;
- после фактического прибытия груза для всех заказчиков строятся фактические накладные и суммы накладных добавляются в баланс заказчика; распечатывается комплект документов;
- при выдаче или отправке товаров заказчику накладная и её товары получает признак “выдано”.

Расчёты с заказчиками

Ответ на вопрос “кто кому должен?”, как всегда, оказывается одним из самых сложных в системах рассматриваемого типа. Основные трудности расчётов связаны с разными валютами каталогов и валютой оплаты заказов, а также текущие колебания курсов валют. Для преодоления этих трудностей приемлемую стратегию можно сформулировать следующим образом:

- каждый заказчик выбирает для себя базовую валюту расчётов – рубли, доллары или евро;
- при оформлении заказов (в валюте каталога) происходит фиксация рублёвого эквивалента на день заказа;

- после проверки поступившего заказчику груза, фактическая накладная строится с отпускными ценами в валюте каталога; затем вычисляются и запоминаются евро и долларовое сальдо; по фиксированной в исходных заказах отпускной цене в рублях заполняется рублёвый эквивалент накладной;
- услуги, предоставленные заказчику (транспортные, программные и др.), учитываются в фактической валюте и переводятся во все три эквивалента;
- поступившая от заказчика оплата в любой из трёх валют переводится во все три эквивалента очередной платёжной записи по курсам дня получения платежа;
- ежедневно для каждого заказчика строится и рассылается документ "Сверка расчётов", отражающий в валюте баланса заказчика остаток средств на дату предыдущей сверки;
- при необходимости администратор системы может изменить базовую валюту заказчика (все необходимые данные в системе есть).

Взаимодействие с заказчиками

Информирование заказчиков о всех стадиях прохождения заказов, возникающих проблемах и состоянии финансового счёта осуществляется следующим образом:

- ежедневно строится и при необходимости рассылается заказчикам документ "Товары в работе";
- по мере поступления и выверки товара заказчикам отправляются фактические накладные;
- ежедневно заказчикам передаются сверки расчётов;
- в случае заказов с безналичной оплатой высылаются предварительные счета с зафиксированными рублёвыми ценами;
- вместе с товарами заказчикам передаются фактические накладные, счёт-фактуры и товарные документы;
- для информирования заказчиков о текущем уровне цен, разработаны клиентские прайс-листы;
- наличие и стоимость товаров на складе фирмы и в магазинах публикуется в Интернете в соответствующих прайс-листах.

Безопасность

Значительный объём товаров, ежедневно проводимых через фирму, требует применения следующих специальных мер, направленных на обеспечение безопасности использования данных:

- защита данных на уровне пользователей;
- фиксация внутреннего сетевого номера компьютера и номера специалиста во всех основных записях базы данных;
- протоколирование всех основных операций;
- невозможность прямого доступа к таблицам базы данных;
- создание перекрёстных отчётов, выявляющих несоответствие ценовых показателей или отклонения от принятой технологии;
- ежедневное копирование базы данных;
- защита от несанкционированного внешнего доступа к локальной сети фирмы.

Структура системы

Основными программными компонентами системы являются:

- администратор,
- учёт доходов и расходов фирмы,
- заказы юридических лиц,
- бухгалтерские документы,
- оптовый склад,
- заказы физических лиц,
- менеджер магазина,
- продавец-консультант,
- кассир,
- программные услуги.

Система работает более пяти лет и доказала свою надёжность и эффективность. Общее количество пользователей более 50, ежедневно обрабатывается более 1000 заказов в день.

ЛИТЕРАТУРА

1. *Краснощёков П.С., Флёров Ю.А.* Иерархия задач проектирования. // Задачи и методы автоматизированного проектирования. М.: ВЦ РАН, 1991, с. 3-23.
2. *Краснощёков П.С., Савин Г.И., Фёдоров В.В., Флёров Ю.А.* Автоматизация проектирования сложных объектов машиностроения.//Автоматизация проектирования, 1996, № 1, с. 5-19.
3. *Вышинский Л.Л., Гринёв И.Л., Демидов А.Ю., Широков Н.И.* Технологии разработки и сопровождения АБС. // Банковские технологии, 1997, № 6, с. 44-49.
4. *Вышинский Л.Л., Гринёв И.Л., Катунин В.П., Лабутин И.В., Флёров Ю.А. Широков Н.И.* Банковские Информационные технологии (части I и II). М.: ВЦ РАН, 1999, 272 с.
5. *Вышинский Л.Л., Гринёв И.Л., Флёров Ю.А., Широков А.Н., Широков Н.И.* Генератор проектов – инструментальный комплекс для разработки «клиент - серверных» систем // Информационные технологии и вычислительные системы. 2003, № 1-2, с. 6-25.
6. *Вышинский Л.Л., Гринёв И.Л., Флёров Ю.А., Широков А.Н., Широков Н.И.* «Автоматизация проектирования прикладных информационных систем» // М.: ВЦ РАН, 2003, 61 с.
7. *David A. Marca, Clement L. McGowan*. SADT: Structured Analysis and Design Technique. McGrow-Hill Book Company, New York, 1988.
8. *Brackett, J., and C. McGowan*: “Applying SADT to Large System Problems”, SofTech Technical Paper TP059, January 1977.
9. SofTech, Inc. “Introduction to IDEF0”, SofTech Deliverable no. 7500-14, September 1979.

10. *Широков А.Н.* Организация сетевого взаимодействия при автоматизации разработки программных комплексов. // М.: ВЦ РАН, 2004, 26 с.

11. *Пярин В.А., Кузьмин А.С., Смирнов С.Н.* Безопасность электронного бизнеса / Под ред. В.А. Минаева. М.: Гелиос АРВ, 2002, 432 с.

ОГЛАВЛЕНИЕ

**Автоматизация проектирования
финансовых информационных систем**

Редактор Н. П. Петрова
Корректура авторов

Подписано в печать 26.10.2004.
Формат бумаги 60×90 1/16
Уч.-изд. л. 6,8. Усл.-печ. л. 8.
Тираж 120 экз. Заказ 42

Отпечатано на ротапринтах в ВЦ РАН
119991, Москва, ул. Вавилова, 40