РОССИЙСКАЯ АКАДЕМИЯ НАУК
ВЫЧИСЛИТЕЛЬНЫЙ ЦЕНТР ИМЕНИ
А.А.ДОРОДНИЦЫНА

АВТОМАТИЗАЦИЯ ПРОЕКТИРОВАНИЯ ИНЖЕНЕРНЫХ И ФИНАНСОВЫХ ИНФОРМАЦИОННЫХ СИСТЕМ СРЕДСТВАМИ ГЕНЕРАТОРА ПРОЕКТОВ

ВЫЧИСЛИТЕЛЬНЫЙ ЦЕНТР РАН
МОСКВА 2010

УДК 519.86

Ответственный редактор
чл.-корр. РАН Ю.А. Флеров

В сборнике статей представлены работы, посвященные описанию опыта использования технологии Генератора проектов, который в течение нескольких лет разрабатывается и используется в Вычислительном центре РАН. Генератор проектов предназначен для автоматизации проектирования клиент-серверных прикладных информационных систем коллективного доступа. Основным достоинством технологии Генератора проектов (ГП) является то, что создание прикладных программно-информационных систем начинается с разработки полноценного проекта, который однозначно описывает все свойства системы и который является исходным документом (совокупностью документов) для автоматической генерации полного программного кода системы. Статьи сборника посвящены, с одной стороны, новым направлениям и задачам, которые решаются для совершенствования технологии ГП, а с другой стороны, описанию прикладных информационных систем, разработанных в последнее время с помощью этой технологии.

Ключевые слова: автоматизация проектирования, информационные системы, коллективный доступ, генерация программ

Работа частично поддержана проектом РФФИ № 10-07-00206а и проектом № 203 Президиума РАН.

Рецензенты: А.М. Попов, А.В. Лотов.

Научное издание

ГЕНЕРАТОР ПРОЕКТОВ - СРЕДСТВО АВТОМАТИЗАЦИИ ПРОЕКТИРОВАНИЯ ПРИКЛАДНЫХ ИНФОРМАЦИОННО-ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Ю.А. Флёров, Л.Л. Вышинский, И.Л. Гринев, А.А. Логинов, А.Н. Широков, Н.И. Широков

Опыт разработки прикладных программ прошедших двух - трех десятилетий состоит в том, что развитие программных средств происходило с неуклонным существенным увеличением и усложнением программного кода. При этом усложнение исходных текстов прикладных программ лишь частично связано с развитием и усложнением математических моделей, положенных в основу решения прикладных задач. В основном, сложность современных программ связана с желанием, а в большей части, с необходимостью обеспечить их работу в современной вычислительной среде, со стремлением использовать все имеющиеся доступные информационные ресурсы, придать программам внешний лоск и «товарный» вид. С одной стороны, это, конечно, улучшает качество программ и создает определенные удобства пользователям, обеспечивает их разумным (часто говорят интеллектуальным) интерфейсом и соответствующей инфраструктурой. А с другой стороны, как правило, конечный практический результат применения этих прикладных программ остается на уровне аналогичных программ, написанных на древнем Фортране или Алголе-60. При этом объем старых программ, решающих те же задачи, был в десятки раз меньше. Таким образом, создание инфраструктуры современных прикладных программ (систем) обходится, как и всякая инфраструктура, очень дорого. Авторы этого сборника во всей полноте испытали подобные трудности, разрабатывая системы автоматизации в различных прикладных областях. В качестве характерного примера можно привести разработку системы АСВР, в создании которой им довелось участвовать. АСВР - это система весовых расчетов, применяемая при проектировании самолетов.

Она была разработана более тридцати лет назад для ЭВМ БЭСМ-6, а затем модифицировалась для VAX/VMS и эксплуатировалась в реальном проектировании много лет. Когда не стало ни БЭСМ-6, ни VAX, к авторам системы обратились с просьбой сделать ее «ремейк». Эта работа была выполнена на новом уровне, новыми средствами, но большинство функций весовых расчетов по просьбе заказчика остались прежними, поскольку они полностью обеспечивали решение тех практических задач, которые перед ней ставилось. Тем не менее объем программного кода новой реализации по сравнению со старой программой вырос более чем в 20 раз. Конечно, эти две версии нельзя сравнить ни по внешнему виду, ни по удобствам работы с ними, но за это пришлось платить существенным увеличением объема исходных текстов программы. Если пользоваться старыми нормами труда разработчиков программ, то для реализации этого ремейка не хватило бы никаких человеко-месяцев. К счастью, в настоящее время и производительность труда программистов тоже существенно изменилась за счет специальных инструментальных средств и новых технологий создания программных систем. При разработке новой версии АСВР использовался инструментальный комплекс Генератор проектов (ГП), что позволило решить задачу за реальное время. Настоящий сборник посвящен описанию опыта использования технологии ГП, которая в течение нескольких лет разрабатывалась и использовалась в Вычислительном центре [1]. Надо сказать, что комплекс Генератор проектов предназначен для автоматизации проектирования и разработки определенного класса прикладных систем, а именно многоуровневых клиент-серверных систем коллективного доступа.

Как уже говорилось, сейчас существует большое количество самых разнообразных инструментальных средств и библиотек, позволяющих писать прикладные программы с современным интерфейсом, с использованием мощных систем управления базами данных, с относительно простым доступом к телекоммуникационным ресурсам. В настоящее время рынок инструментальных программных средств (CASE-средств) насчитывает сотни наименований. Сделать какой-либо систематический обзор этих средств в сборнике не представляется возможным, поскольку слишком велико разнообразие задач

автоматизации программирования, которые в них ставятся. Однако основные из них следует упомянуть.

Одно из основных направлений связано с разработкой интегрированных инструментальных средств вокруг систем управления базами данных. Наиболее известным и наиболее мощным CASE-средством является **Oracle Developer Suite** фирмы ORACLE. Эта интегрированная среда обеспечивает автоматизацию разработки и поддержки прикладных информационных проектов, ориентированных на создание, ввод, хранение, модификацию, вывод и визуализацию сложных структур данных. Oracle Developer Suite обеспечивает управление данными, программирование или включение в среду различных стандартных или пользовательских бизнес-процедур, предоставляет все необходимые средства инфраструктуры проекта. Oracle Developer Suite включает следующие основные компоненты: Oracle Designer - проектирование баз данных и приложений, Oracle Developer - разработка приложений и бизнес процедур, Oracle Forms Developer - разработка пользовательского интерфейса на основе экранных форм, Oracle Reports Developer - разработка отчетов, Oracle Warehouse Builder - проектирование хранилищ данных, Oracle Discoverer - разработка аналитических приложений. Помимо перечисленного набора программных продуктов в составе Oracle Developer Suite входит набор компонентов, утилит и интерфейсов для организации работы с XML-документами, для моделирования классов и рабочих процессов, для генерации результирующего программного кода. В Oracle Developer Suite предусмотрен репозиторий, как единый источник метаданных приложений, обеспечивающий эффективную параллельную работу коллективов разработчиков, которые могут анализировать и контролировать зависимости между объектами, использовать общие компоненты, управлять конфигурациями, обновлять версии и вести их архив.

Неоспоримые достоинства и универсальность Oracle Developer Suite часто переходят в основной ее недостаток – ресурсоемкость, высокая стоимость, сложность внедрения и использования. Такую особенность приходится учитывать, когда идет речь о разработке относительно небольших и средних пользовательских приложений, в которых реально необходим

лишь небольшой процент возможностей столь мощной инструментальной среды. В принципе, существуют более простые и относительно недорогие средства примерно с теми же функциями. Например, **Uniface** - продукт фирмы Compuware (США), **Vantage Team Builder** и другие. Достаточно много CASE-средств связано с графическим проектированием прикладных программ, основанных на построении функциональных и информационных моделей в виде диаграмм потоков данных и диаграмм "сущность-связь". Наиболее популярны среди них система **Rational Rose** компании Rational Software Corp, система **Silverrun** американской фирмы Computer Systems Advisers, Inc. (CSA) и ряд других систем. Одним из немногих отечественных CASE-средств функционального моделирования проектируемых информационных систем является система **CASE.Аналитик**. Ее основные функции состоят в построении и редактировании диаграмм потоков данных, анализе диаграмм и проектных спецификаций на полноту и непротиворечивость, получении разнообразных отчетов по проекту, генерации макетов документов.

Наиболее близкий к технологии Генератора проектов подход реализован в инструментарии, разработанном фирмой JYACC (США) под названием **JAM (JYACC Application Manager).** Он предназначен для создания приложений в информационных системах архитектуры "клиент-сервер", построенных на базе реляционных СУБД.

Генератор проектов как система автоматизации разработки клиент-серверных программных комплексов, была впервые создана в начале 90-х годов, в качестве попытки автоматизировать написание однотипных программных компонент, возникших тогда в работе над банковскими системами. За прошедшие почти два десятилетия архитектура ГП, язык описания проектов и собственно концепция проекта были существенно видоизменены. Однако основные концепции ГП, заложенные в 90-х гг., остались неизменными.

1. Разработка прикладных программно-информационных систем в рамках технологии ГП основывается на "проектном подходе", т.е. ведется на основе формального описания проекта,

являющегося исходным документом для автоматической генерации полного программного кода системы.

2. Разрабатываемые в рамках технологии ГП прикладные системы обеспечиваются эффективными средствами сопровождения, то есть достаточно простой процедурой внесения исправлений и развития программ в процессе функционирования. Эффективность этих процедур гарантируется сквозной технологией разработки от проекта до конечного исполняемого кода.

3. В рамках технологии ГП обеспечивается возможность разработки прикладных систем многоуровневой клиент-серверной архитектуры с использованием реляционных (а теперь и сетевых) баз данных со сложным пользовательским и межпрограммным интерфейсом.

4. В рамках технологии ГП обеспечивается высокая степень надежности и информационной безопасности; в разрабатываемых прикладных системах допустима интеграция с внешними средствами кодирования и декодирования информации для транспортировки по публичным телекоммуникационным каналам.

5. Прикладные программные комплексы в рамках технологии ГП разрабатываются как автономные системы и не требуют для своей работы дорогостоящих программных продуктов (кроме использующихся СУБД и общесистемного обеспечения).

6. Разрабатываемые в рамках технологии ГП прикладные системы допускают масштабирование и портирование на различные вычислительные платформы и СУБД.

«Проектный подход» в технологии ГП подразумевает определенную модель прикладной системы. В первой реализации ГП модель проекта прикладной системы предполагала наличие одного сервера и нескольких клиентских модулей. Клиентский модуль – это программа с пользовательским интерфейсом в виде экранных форм, способная выполнять по команде пользователя запросы к серверу и выдавать результаты этих запросов в виде совокупности таблиц. Модель сервера представляла собой совокупность именованных запросов, каждый с фиксированным

комплектом входных и выходных параметров. Комплект входных параметров передавался от клиента. Выходные параметры запросов могли быть в виде одного кортежа, как входные, а также в виде совокупности, т.е. таблицы. В модели предполагалось, что сервер работает с одной базой данных. Структура базы данных описывалась, как часть модели. Это позволяло генерировать утилиты для ведения базы данных, такие как создание пустой базы, автоматизированная процедура обновления метаданных базы при установке новой версии проекта и пр.

В проекте предполагалась значительная доля ручного программного кода на языке Си: в первую очередь программы простых запросов и ручные вставки в клиентские процедуры. Был также разработан редактор экранных форм, в котором можно было размещать произвольные тексты, именованные поля и таблицы с именованными полями.

Несмотря на достаточно примитивную модель проекта, описанная инструментальная система была использована для разработки многих реальных проектов и показала свою эффективность при проектировании и разработке программ, но особенно при эксплуатации, сопровождении и модернизации сгенерированных программ. Сопровождение - это особенно важная сторона жизненного цикла прикладных систем. Системы, живущие 3-5 и более лет, время от времени нуждаются в частичном изменении и структур данных, и бизнес–процедур, и интерфейсов. Разработчики прикладных систем знают насколько болезненно проходят любые вмешательства в работающую программу. Одно из основных достоинств Генератора проектов состоит в том, что изменения вносятся не в программный код, а в описание проекта, при этом программный код модифицируется автоматически, а для проведения необходимых изменений в структурах проекта автоматически создаются специальные утилиты. Такой подход позволяет не нарушать целостность работающих систем.

Генератор проектов не ориентирован на какую-то конкретную область применения и может быть использован при разработке широкого класса прикладных информационно-вычислительных систем. Реально ГП используется с начала девяностых годов. Первые версии использовались для

автоматизации разработки банковских систем. Были разработано несколько систем для Центрального банка и Сбербанка России. При разработке этих систем, ГП постоянно расширялся и по своим внутренним языковым возможностям, и по интерфейсам, и по классу решаемых задач. В дальнейшем ГП использовался для построения приложений, выходящих за банковскую сферу. Ниже перечислен ряд систем, разработанных средствами Генератора проектов в разные годы:

- 1992 г.: Система АВИЗО для ГУ ЦБ РФ по г. Москве.
- 1993 г.: Система "АС МБР" для ГУ ЦБ РФ по г. Москве.
- 1993 г.: Проект платежной системы на основе смарт-карт для Сбербанка России.
- 1994 г: Проект интегрированной системы автоматизации банковской деятельности.(Заказчик Владимирский банк СБ РФ).
- 1996 г.: Система ГАМБИТ для Башкирского банка СБ РФ ([2]).
- 1998 г.: Система КЛИРИНГ для КБ «Роспромбанк». ([3])
- 1999 г.: Система аудита биллинга сотовых операторов. (Заказчик: Группа страховых компаний НАСТА).
- 1999 г.: Пилот-проект «Electronic Transcribing System». (Заказчик г.: «BrainStorm Engineering, LLC» , США). Прототип многопользовательской автоматизированной системы для распределенной обработки медицинской информации через Internet.
- 1999 г.: Проект электронной коммерции eCommerce на базе микропроцессорных карт СБЕРКАРТ Сбербанка России.
- 2002 г.: Система Duplet для обеспечения Интернет торговли и проведения некоторых банковских операций в режиме реального времени на базе микропроцессорных карт СБЕРКАРТ Сбербанка России ([4]).
- 2003 г.: Проект системы мобильного банкинга. Заказчик г.: ЗАО «СмартКарт Сервис» ([5]).
- 2005 г.: Система MassPay для приема широкого спектра платежей населения (коммунальные, налоговые, оплата товаров и услуг и т.п.) на банковских устройствах самообслуживания - банкоматах. Заказчик : Сбербанк России ([6]).
- 2005 г.: Биллинговая система «Енисей». Заказчик г.: Восточно - Сибирский банк Сбербанка России.

- 2005 г.: АСБУ - Автоматизированная система бюджетного управления корпорацией. Заказчик г.: ОАО ТВЭЛ.
- 2007 г.: Автоматизированная система весовых расчетов (АСВР) Заказчик г.: КБ им. П.О. Сухого.
- 2010 г.: Комплексная программа управления инженерными расчетами – КПИР. Заказчик г.: КБ им. П.О. Сухого.

Во всех перечисленных проектах, разработчики получали существенную экономию трудозатрат на всех этапах жизненного цикла систем: проектирования, создания готовых программных продуктов, сопровождения и модификаций. Перечисленные проекты, разработанные в разные годы, стимулировали включение в Генератор проектов все новых и новых технических и технологических средств. Как и любая востребованная и работающая система, Генератор проектов много раз модифицировался, постоянно расширялись его возможности, совершенствовалась архитектура разрабатываемых прикладных программ. Поэтапно осуществлялся переход на оконный интерфейс. Первая попытка использования в ГП оконного интерфейса основывалась на применении экранных форм. Однако реальная практика использования Генератора проектов при разработке прикладных программ и возросшие требования заказчиков к пользовательскому интерфейсу (самое распространенное требование - «чтоб было красиво») заставили модифицировать Генератор проекта и изменить модель проекта. Для этого был разработан специальный **базовый язык** описания проекта. Изначально базовый язык предназначался как подмножество языка описания проектов, как вспомогательное средство для замены Си-кода в составе проекта. На практике получилось, что Си-программы в проектах остались только там, где без них нельзя обойтись принципиально – для обеспечения интерфейса со сторонними библиотеками. В базовом языке реализован строгий контроль использования типов, значительно более сильный, чем в Си. Кроме того производится тщательный контроль используемости самых разнообразных объектов модели. Это свойство чрезвычайно полезно при работе с большими проектами. Опыт поэтапной реализации этих средств показал, что при отсутствии должного контроля разработчик не в состоянии

уследить за корректностью модели, она начинает обрастать излишествами. Проблема управления памятью в базовом языке как таковая отсутствует, так как в нем нет явных динамически создаваемых объектов вроде экземпляров структур/классов языков Си/С++. Вместо этого в языке есть понятие “закрываемого” типа данных, экземпляр которого после использования следует явно закрыть во избежание утечки памяти. Переменные таких типов допустимо описывать только в строго определенных контекстах, гарантирующих корректное их закрытие.

В базовом языке принята простая концепция реакции на ошибки. Любая процедура (встроенная процедура) языка может быть завершена неявно при обнаружении ошибки в вызываемой другой процедуре или явно оператором error “текст ошибки”.

С появлением достаточно универсального базового языка появился соблазн генерировать отдельные компоненты проекта не сразу на Си, а сначала на базовый язык, а уж затем с базового языка генерировать на Си. При этом существенно сокращается код ГП, так как исчезает необходимость отслеживать присущие языку Си нюансы адресации разных типов (расстановка символов &*.->). Вызовы процедур становятся нагляднее и, самое главное, генерируемый код на базовом языке в отличие от Си допускает более тщательный контроль на избыточность.

Существенным шагом в развитии ГП явилось введение нового понятия – структурированного документа, как типа данных. Хотя, это понятие не совсем новое для нашей технологии. Структурированный документ как тип данных имеет давнюю историю. Изначально такие структуры были предложены в концепции сетевых баз данных (стандарт CODASYL) на рубеже 70-80 годов. Сейчас эта концепция в угоду массовой культуре программирования благополучно забыта. Коллектив разработчиков Генератора проектов использовал стандарт CODASYL в своих давних разработках в 80-х годах в программных продуктах для машиностроительной отрасли. Потом эта конструкция была временно заброшена в связи с использованием промышленных реляционных баз данных.

Вторая жизнь сетевых структур началась в середине 90-х годов, когда сложность модели проекта, реализуемого ГП, превысила некий пороговый уровень. Это потребовало новых

технологий в разработке самого Генератора проектов. Использование стандартных возможностей языка Си для реализации модели проектов оказалось неэффективным. Потребовалось автоматизировать сам процесс разработки генератора. Отметим, что в текущей версии ГП модель проекта представляется структурированным документом, в котором имеется 140 типов записей и 483 типа наборов (связей) между ними. Если учесть, что для каждой записи и для каждого набора в общем случае может быть использовано порядка 30-40 видов встроенных операций, то можно себе представить сложность задачи.

Переход к новой версии ГП как раз и ознаменовался легализацией структурированных документов как одного из стандартных типов, используемых при описании модели проекта. Базовый язык ГП как раз и разрабатывался для удобного использования документов в программах проекта. Каждому типу записи и набора в соответствии с описанием документа в языке автоматически становились доступными встроенные процедуры манипулирования этими объектами. Использование структурированного документа позволило решить следующие задачи.

1. Построение сложных структурно-параметрических моделей широкого класса предметных областей с иерархической и сетевой организацией любой сложности.
2. Построение протокола передачи данных от клиента на сервер и обратно на основе структурированных документов. Это существенно повысило изобразительную мощь модели проекта.
3. Структурированный документ стал основой для описания оконного интерфейса.
4. Была разработана специализированная форма хранения данных в виде структурированных документов, которые хранятся не в оперативной памяти, а в файлах. Это позволило возродить забытую концепцию сетевых баз данных, квалифицированное использование которых в некоторых случаях помогло достичь высокой скорости обработки информации.

Первый вариант базового языка предусматривал единое пространство имен для типов, процедур, встроенных процедур. Для обеспечения уникальности использовался механизм

префиксов и суффиксов разного рода. Не всегда это было интуитивно понятно, и работа с громоздкими обозначениями была чрезвычайно неудобной. А кроме того, были определенные неудобства при коллективной разработке проектов. Для решения этих двух внешне малосвязанных проблем было проведено серьезное изменение синтаксиса базового языка. В языке было введено понятие пакета. Пакет - это снабженная уникальным в проекте именем совокупность основных сущностей языка: типы, константы, процедуры. Пакеты могут быть разных видов (на данный момент их насчитывается более полусотни). Вообще, после введения пакетов, корневой файл описания проекта теперь содержит кроме заголовочной и вспомогательной информации упорядоченный перечень пакетов разного типа. Вся детальная информация вынесена в файлы пакетов.

В составе ГП имеется совокупность так называемых системных пакетов. Системными они называются потому, что они не описываются разработчиком, а поставляются в составе генератора. В составе системных пакетов есть большое количество пактов типа syspackage. В таких пакетах реализованы многочисленные полезные программы, используемые в макросах или просто рассчитанные на применение в проектах. Это работа с файлами, сетевыми сокетами, лексический разбор, интерфейсы с операционной системой, с графической системой, работа со строками, реализация http-протокола, формирование html-разметки, работа с изображениями, перекодировки и много-много других. Системными пакетам являются оконные классы пользовательского интерфейса, драйверы баз данных. Большинство макросов (системные и серверные администраторы, аудиты, cgi-агенты) тоже являются пакетами, которые создаются генератором.

Благодаря новым мощным средствам описания в новой модели Генератора проектов обеспечивается весьма гибкая структура проекта, которая предусматривает наличие произвольного количества программных компонент - серверов, клиентских приложений с оконным интерфейсом, утилит с вызовом из командных строк, динамических библиотек. В новой модели допускается возможность описания в сервере

произвольного количества портов, web-портов, баз данных. Кроме того, в новой модели допускается возможность использования клиентских подключений не только из приложений, а также из утилит и других серверов и допускается возможность работы с базами данных из утилит и приложений, а не только из серверов.

Несмотря на все введенные новшества в ГП, растущие потребности в новых информационных проектах ставят новые задачи и в связи с этим Генератор проектов требует включение в свой арсенал новых современных средств автоматизации разработки программных систем. Ниже кратко перечислены некоторые задачи, над которыми работают в настоящий момент разработчики Генератора проектов.

Одной из острых проблем при разработке многопользовательских транзакционных клиент-серверных систем является повышение их эффективности. Одним из путей повышения эффективности является построение специальных архитектурных решений, схем взаимодействия серверов с автоматическим распараллеливанием и разделением различных типов запросов к базам данных (быстрых и медленных, типовых и уникальных, массовых и одиночных и т.п.) по разным каналам и разным серверам. Этому вопросу посвящена статья «Многоуровневые структуры в клиент-серверных информационных системах» в настоящем сборнике.

Вообще, вопрос разделения функций клиент-серверных систем на транзакционные (быстрые и массовые on-line функции) и аналитические (off-line функции) весьма важен при разработке автоматизированных систем. Особенно остро стоит эта проблема в финансовых приложениях, в базах данных которых скапливаются большие объемы данных о текущих финансовых потоках. Эта информация является единственной достоверным источником для финансового анализа и планирования деятельности предприятий, но она по многим причинам является труднодоступной из-за неправильной организации хранения и использования. Эта проблема обсуждается в статье «OLTP и OLAP технологии при разработке учетно-аналитических информационных систем».

Традиционно, при разработке пользовательских приложений стоит вопрос экранного интерфейса. Поэтому

разработка новых средств представления информации всегда рассматривается как одна из важных задач для обеспечения возрастающих требований заказчиков. Вопросам разработки новых оконных типов в рамках Генератора проектов посвящена статья «Расширение класса моделей пользовательского интерфейса в Генераторе проектов».

Остальные статьи сборника представляют различные прикладные системы, разработанные средствами Генератора проектов.

Литература

1. *Л.Л. Вышинский, И.Л. Гринев, Ю.А. Флеров, А.Н. Широков, Н.И. Широков.* Генератор проектов – инструментальный комплекс для разработки «клиент - серверных» систем // Информационные технологии и вычислительные системы. 2003, № 1-2. С. 6-25.
2. *Л.Л. Вышинский, И.Л. Гринев., В.П. Катунин, И.В. Лабутин, Ю.А. Флеров, Н.И. Широков.* Банковские Информационные технологии (части I и II) М.: ВЦ РАН, 1999. 272 с.
3. *Л.Л. Вышинский, И.Л. Гринев, Н.И. Широков, В.И Щедров.* Система урегулирования задолженностей // Автоматизация проектирования финансовых информационных систем М.: ВЦ РАН, 2004. С. 110-118.
4. *И.Л. Гринев, А.А. Логинов, Н.И. Широков*. Система ИНТЕРНЕТ-платежей // Автоматизация проектирования финансовых информационных систем // М.: ВЦ РАН, 2004. С. 102-109.
5. *И.Л. Гринев, А.А. Логинов, А.Н. Широков, Н.И. Широков* Система мобильного банковского обслуживания // Автоматизация проектирования финансовых информационных систем М.: ВЦ РАН 2004. С. 95-101.
6. *И.Л. Гринев, А.А. Логинов, А.Н. Широков, Н.И. Широков* Система банковского самообслуживания // Автоматизация проектирования финансовых информационных системМ.: ВЦ РАН 2004. С. 87-94.

ТРАНЗАКЦИОННЫЕ И АНАЛИТИЧЕСКИЕ ПОДСИСТЕМЫ В БАНКОВСКИХ ПРОГРАММНЫХ КОМПЛЕКСАХ

И.Л. Гринёв

Применение инструментального комплекса Генератор проектов позволило в короткие сроки реализовать и внедрить ряд автоматизированных банковских систем (АБС). АБС России в начале 21-го века испытали серьезное давление со стороны розничного бизнеса. В результате наряду со всеми традиционными задачами банковской автоматизации появилась новая проблема, связанная с бурным ростом объема массовых операций клиентов. Серьезной предпосылкой такого роста, безусловно, стало широкое распространение банковских пластиковых карт, что, в свою очередь, привело к лавинообразному увеличению количества электронных платежных транзакций, крайне удобных для клиентов банков, позволяющих выполнять даже небольшие (несколько рублей) операции без применения наличных.

При этом современные АБС должны не только справляться с последствиями такого бума, но и предоставлять возможности для продолжения интенсивного роста. Таковы обязательные требования к АБС в условиях усиливающейся конкуренции на рынке розничного банковского обслуживания.

Учетные функции АБС по-прежнему остаются сегодня обязательными и крайне важными, но они уже недостаточны для эффективного функционирования современного банка. Расширение требований к функциональности АБС привело к изменению архитектуры программных банковских комплексов, а также к появлению новых подсистем, обеспечивающих решение вновь возникающих задач в области розничного банковского обслуживания.

«Быстрые» и «медленные» процедуры в АБС

С точки зрения оперативной деятельности банков АБС должны обеспечивать обработку весьма большого количества операций в сутки в режиме on-line (24x7x365). При этом для удобства клиентов банки используют различные каналы продаж

банковских продуктов, традиционные, (обслуживание клиента «у окна» банковским операционистом), и удаленные (сети устройств самообслуживания, Интернет-банк, GSM-банк и пр.)

Для обеспечения возможности функционирования таких сложных комплексов банковская автоматизированная система должна иметь возможность очень быстрого выполнения типовых операций обработки электронных банковских документов. Эта обработка заключается, как правило, в контроле реквизитов документов с последующей их записью в Базу данных АБС, а также в выполнении ряда действий по модификации некоторых других записей в Базе данных. Традиционно этот круг задач, решаемых в учетно-аналитических автоматизированных системах, относят к подсистемам класса OLTP (Online Transaction Processing).

Чем удобнее организовано обслуживание клиентов, тем активнее они используют разнообразные каналы приобретения банковских продуктов и услуг и тем больше операций обрабатывают АБС. В настоящее время для среднего банка характерным показателем можно считать сотни тысяч и миллионы розничных операций в сутки, выполняемых в режиме on-line. К таким операциям можно отнести коммунальные платежи, оплату телекоммуникационных услуг, платежи по ссудам, штрафы, налоги, оплату авиационных и железнодорожных билетов и т.д.

Программные процедуры в АБС, обеспечивающие обработку таких операций, и должны быть самыми «быстрыми». На сегодняшний день АБС современного розничного банка в пиковые моменты обеспечивают обработку сотен и тысяч электронных транзакций в минуту. Функционирование таких высокопроизводительных банковских систем даже за небольшие промежутки времени приводит к появлению в банковских Базах данных очень больших объемов информации, а многолетние архивы подобных автоматизированных комплексов содержат поистине огромные массивы данных. С одной стороны, эти данные (а это не только данные по выполненным операциям, но и многолетние структуры нормативно-справочной информации - НСИ, информация о клиентах и др.) представляют для Банка реальную ценность, поскольку позволяют ставить и решать разнообразные аналитические, статистические, управленческие

задачи. В широчайший перечень подобных задач можно включить контроль и управление эффективностью внедрения банковских проектов (продуктов), персонифицированный подход к активному предложению банковских продуктов клиентам, оперативное построение управленческой отчетности для руководителей различного уровня, подготовка статистических отчетов и т.п.

Программные процедуры АБС, решающие эти задачи, вполне могут быть реализованы как гораздо более «медленные», чем транзакционные процедуры, поскольку зачастую минуты, и даже часы, затраченные на их исполнение, являются вполне приемлемыми показателями. Подсистемы, используемые для решения этих задач, как правило, относят к классу OLAP (Online Analytical Processing).

При этом необходимо отметить, что большой объем Базы данных АБС и необходимость выполнения «медленных» процедур негативно сказываются на скорости выполнения в системе задач класса OLTP.

Организация современных АБС

С учетом разных требований к оперативности исполнения задач рассматриваемых классов вполне логичным является разделение соответствующих программных процедур между подсистемами OLTP и OLAP в рамках единой АБС. Кроме того, целесообразным видится и распределение данных между подсистемами, даже если это приведет к их частичному дублированию. Так, например, «быстрая» процедура обработки платежа клиента с его текущего счета, а также архив выполненных операций дня, очевидно, должны быть включены в состав OLTP-подсистемы АБС. И наоборот, процедура построения выписки по счету клиента за прошлые периоды и электронный архив операций за несколько лет являются частью OLAP-подсистемы.

Но OLAP-подсистемы должны быть в достаточной степени интегрированы в АБС. Это означает, что оперативные данные из OLTP-подсистемы должны в автоматическом режиме регулярно поступать в OLAP-подсистему. Также обе подсистемы могут в ряде случаев использовать общие источники НСИ.

Кроме того, наличие в АБС разнотипных подсистем не должно усложнять деятельность ее пользователей (как

сотрудников банка, так и его клиентов), поэтому при выполнении сложных функций, предполагающих взаимодействие OLTP и OLAP подсистем, это взаимодействие должно быть автоматическим и оперативным, насколько того требует сама функция. Можно сказать, что обе подсистемы функционируют в едином информационном пространстве, но при этом оно довольно сложно организовано с учетом необходимости разделения функций между различными частями единой АБС.

Сложная структура информационного обеспечения системы, в свою очередь, ведет к усложнению ее программного обеспечения, поскольку именно программная реализация обеспечивает единство информационного пространства с точки зрения пользователя.

Клиентоориентированный подход к проектированию АБС

Современные розничные АБС, перед которыми ставятся задачи по обслуживанию как можно большего количества клиентов, а именно – обработки как можно большего количества клиентских операций в единицу времени, должны быть построены соответствующим образом. В логической схеме такой АБС центральной компонентой видится программная подсистема, которая выполняет следующие группы функций:

- Ведение реестра клиентов.
- Ведение реестра электронных шаблонов операций для каждого клиента, стандартных и индивидуальных (профиль клиента).
- Идентификация и аутентификация клиента при попытке подключения к системе.
- Поддержка различных каналов доступа клиентов к ресурсам системы (автоматизированные рабочие места сотрудников банка, сети устройств самообслуживания, Интернет, GSM-сети и т.д.).
- Обеспечение информационных обменов с различными внешними программными комплексами (подсистемы АБС, подсистемы поставщиков услуг и т.д.) в режимах on-line и off-line для целей обработки клиентских операций и для обеспечения связанных с ними расчетов.

Именно наличие в составе АБС программной компоненты с такой функциональностью позволяют клиентам банка в

защищенном режиме, удобным способом, практически в любое время и в различных географических точках выполнять широчайший спектр банковских операций. При этом уровень сервиса, по сути, зависит от разнообразия и оперативности способов, которыми клиенты могут передать своему банку (банкам) поручения на использование собственных средств.

Очень схожие по основным функциям программно-технические комплексы уже многие годы в своей деятельности используют, к примеру, операторы сотовой связи. Они называются биллинговыми системами. Именно биллинговая подсистема распознает абонента, выполняющего телефонный звонок или отправляющего сообщение; контролирует возможность оказания услуги (звонок, SMS, передача данных); тарифицирует оказанную услугу; ведет лицевой счет абонента, учитывая на нем стоимость оказанных услуг; учитывает выполненные клиентом пополнения счета, его задолженности; готовит для клиентов счета на оплату услуг и т.д.

Новый класс программных подсистем АБС, основную функциональность которых мы рассмотрели выше, принято называть банковскими биллингами. Информационным фундаментом банковской биллинговой системы является реестр клиентов банка, который используется при любой попытке доступа к ресурсам системы, позволяет в удобном виде и через удобный канал предоставить возможность клиенту выполнять операции из его профиля, многие из которых имеют индивидуальные настройки. Именно индивидуальные настройки операций позволяют клиентам банка существенно экономить время на их дальнейшем выполнении. Так, например, десятки реквизитов электронного платежного документа могут быть заданы заранее и будут храниться в профиле клиента, а плательщик при выполнении операции должен будет указать минимум информации – сумму платежа.

Проблемы построения банковских биллинговых систем

Можно предположить, что сама по себе потребность в появлении банковских биллинговых подсистем диктуется современным состоянием банковской автоматизации в России. Нынешний период можно назвать переходным, когда постепенно

начнут появляться АБС нового поколения, и та функциональность, которая требуется для эффективной организации розничного банковского обслуживания (и в первую очередь – самообслуживания) будет органично включена в новые системы. Важными объективными предпосылками такого развития являются и новейшие системотехнические принципы построения больших учетно-аналитических систем, такие как «облачные вычисления», SOA-архитектуры и т.д.

Однако на широкомасштабное практическое применение этих принципов в реальных АБС уйдут годы, а решение ряда конкретных задач розничного банковского обслуживания уже не терпит отлагательства. В данной ситуации внедрение в рамках существующих АБС биллинговых подсистем представляется, пусть временным, но весьма полезным этапом, на котором, кстати, довольно удобно было бы оценить практическую ценность новых системотехнических идей и удобство передовых интеграционных платформ, таких, например, как IBM WebSphere.

Дело в том, что банковский биллинг, будучи выделен в отдельную подсистему АБС, неминуемо столкнется с рассмотренной нами выше проблемой «сосуществования» в рамках единого программного комплекса «быстрых» и «медленных» программно-технологических процедур. Поскольку на текущем этапе развития банковских информационных технологий биллинговая подсистема становится централизующей многоканальной компонентой, обеспечивающей слаженное взаимодействие десятков и сотен автоматизированных подсистем, она же становится и «узким местом». Такая ситуация в условиях увеличения нагрузки (рост количества операций клиентов) может привести к нестабильной работе всей АБС.

Решение данной проблемы, по всей видимости, необходимо искать в декомпозиции самой биллинговой подсистемы. Предлагается начать разделение программных систем на OLAP и OLTP подсистемы с банковского биллинга, что, несомненно, будет проще сделать, поскольку биллинговые компоненты являются наиболее молодыми продуктами и должны быть более гибкими и технологичными, чем другие компоненты АБС, многие из которых на сегодняшний день уже принципиально невозможно модифицировать.

По всей видимости, при проектировании банковского биллинга в каждом конкретном случае, в зависимости от масштабов банковского бизнеса и ИТ-ландшафта организации, архитектура разрабатываемой подсистемы может быть уникальной. Вместе с тем полезно рассмотреть некоторые типовые подходы в проектировании и разработке, которые позволят в результате получить работоспособный программный комплекс, способный надежно функционировать при требуемых уровнях нагрузки.

Один из этих важнейших принципов уже сформулирован выше и заключается в том, что критичные ко времени выполнения некоторых функций программные системы полезно разделять на подсистемы, выполняющие «быстрые» и «медленные» процедуры.

Предположим, что декомпозиция биллинговой подсистемы будет вестись в условиях реструктуризации АБС, основанной на внедрении новой интеграционной платформы, включающей в свой состав некую общую шину передачи данных. В этих условиях вся совокупность программных подсистем банка должна быть представлена в виде набора программных процедур (сервисов), взаимодействующих между собой единообразно через общую шину.

В общем случае любая функция АБС будет обеспечиваться взаимодействием конкретного набора сервисов в определенной последовательности через общую шину передачи данных. Режимы взаимодействия этих сервисов могут быть как синхронными, так и асинхронными, в зависимости от сценария выполнения конкретной функции. Важно отметить, что в соответствии с предлагаемой нами моделью, некоторые из сервисов будут «быстрыми», а другие – «медленными».

Именно таким образом должна быть реструктурирована, либо построена заново и биллинговая подсистема АБС. Она будет представлять собой набор «быстрых» и «медленных» сервисов (программных процедур) ограниченной функциональности, которые будут взаимодействовать между собой по определенным правилам для целей выполнения биллинговых функций.

При этом некоторые из сервисов могут быть довольно сложно организованы, а их программная реализация будет включать в себя не только прикладные программы, но и

автономные базы данных, интерфейсы с другими программными комплексами и т.д. Например, к «быстрым» сервисам обязательно будет относиться процедура проверки реквизитов и записи в базу данных электронного документа на списание средств со счета клиента в пользу поставщика товара/услуги. Как «медленный» сервис можно будет реализовать процедуру построения электронных документов, необходимых для осуществления расчетов (в т.ч. и межбанковских) по результатам выполненной розничной операции.

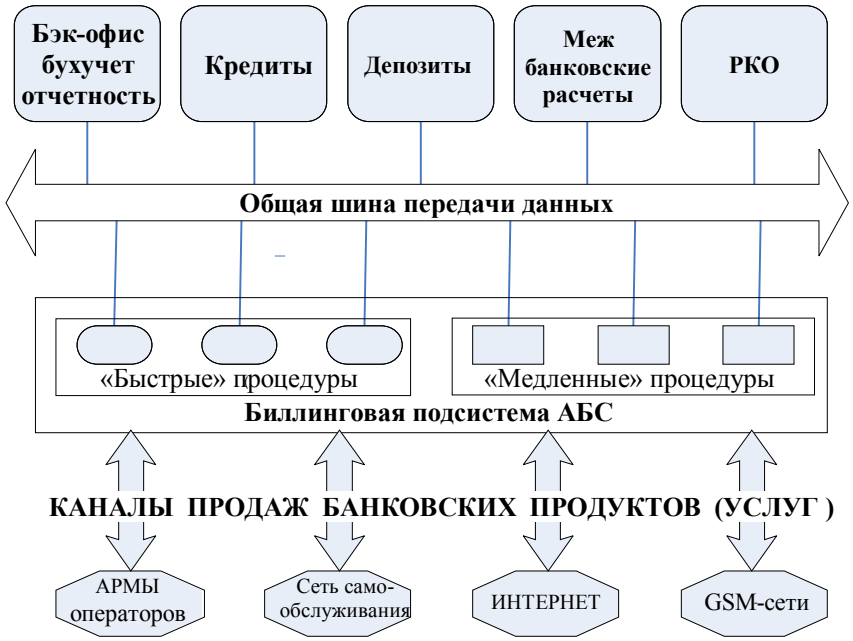


Рис. 1. Декомпозиция биллинговой системы

На рис.1 биллинговая подсистема АБС представлена как набор «быстрых» и «медленных» процедур, которые взаимодействуют через общую шину данных как между собой, так и с другими подсистемами (сервисами) АБС.

Вместе с тем на первых этапах реструктуризации АБС можно провести декомпозицию биллинговой подсистемы на

отдельные процедуры с учетом рассмотренных в статье принципов, но интерфейсы взаимодействия этих процедур могут быть реализованы традиционными способами без использования новейших интеграционных платформ. И даже такая реструктуризация может дать серьезный положительный результат, если удастся организовать независимое и асинхронное функционирование различных процедур.

Автоматизация проектирования современных АБС

Традиционно в ВЦ РАН большое внимание уделяется технологиям автоматизации программирования при создании больших и сложных программных систем. Применение Генератора проектов позволило реализовать ряд проектов в области розничного банковского обслуживания, в короткие сроки отладить и запустить их в промышленную эксплуатацию, успешно сопровождать в течение многих лет. В то же самое время и сам Генератор проектов является программным продуктом, который постоянно видоизменяется в связи с растущими требованиями к проектируемым автоматизированным системам.

Потребность в создании биллинговых подсистем АБС, а самое главное – жесткие требования к их производительности и надежности привели к необходимости доработки инструментария разработчика. Важным моментом в развитии Генератора проектов стало дальнейшее расширение возможностей описания модели проектируемого программного комплекса. Теперь в описании проекта возможно описать произвольное количество автономных подсистем с собственными базами данных в едином контуре информационной безопасности со всей системой в целом. Также существенно расширены возможности формализованного описания интерфейсов взаимодействия, как различных подсистем единого программного комплекса, так и интерфейсов с внешними автоматизированными подсистемами. Эти интерфейсы могут быть описаны в модели проекта с учетом различных, применяемых в настоящее время, механизмов: обмен файлами в режиме off-line, синхронные и асинхронные запросы в режиме on-line, Web-сервисы и т.д.

МНОГОУРОВНЕВЫЕ СТРУКТУРЫ В КЛИЕНТ-СЕРВЕРНЫХ ИНФОРМАЦИОННЫХ СИСТЕМАХ

А.А. Логинов, А.Н. Широков, Н.И.Широков

В процессе разработки сложных банковских систем были апробированы различные архитектуры программных комплексов, связанные с модификациями классической трехуровневой схемы клиент-серверной системы (см. рис. 1).

Рис. 1. Классическая трехуровневая архитектура клиент-сервер

Необходимость модификации классической архитектуры была связана с проблемами повышения эффективности классической схемы трехуровневого клиент-сервера и особенностями практической реализации данной схемы в Генераторе проектов. Ниже рассмотрены варианты модификаций трехуровневой архитектуры.

Схема с мультиплексированием

Как правило, к реализации транзакционных систем предъявляется ряд требований и зачастую одним из наиболее критичных требований является ограничение сверху на значение времени отклика прикладного сервера системы во время проведения операции. Ограничение на время отклика может быть выражено и в косвенном виде через требуемое количество операций за определенный интервал времени. В реальных системах на время отклика влияет большое количество факторов. Данные факторы могут определяться как ограничениями программно-аппаратного обеспечения, например скорость передачи данных по сетевым каналам или скорость взаимодействия с дисковой подсистемой, так и временными параметрами прикладной обработки запросов прикладным сервером.

В ряде проектов при разработке транзакционных подсистем пришлось решать проблемы, связанные с факторами

второго типа. В одном из проектов по условиям задачи один из прикладных серверов системы должен был обрабатывать как многочисленные запросы транзакционного цикла (быстрые запросы), так и запросы по построению аналитических отчетов на основе проведенных операций за задаваемый период времени (медленные запросы). Проблема заключалась в том, что время выполнения медленных запросов на несколько порядков превышало требуемое максимальное время отклика. Вызванная обработкой медленных запросов задержка приводила к накоплению большой очереди запросов и временной неработоспособности прикладного сервера. В силу некоторых факторов прямая оптимизация реализации бизнес - логики не могла сократить время обработки проблемных запросов до приемлемого уровня.

Кардинальным решением задачи явилась трансформация первоначально планируемой схемы (по сути – классический клиент-сервер) в схему, представленную на рис. 2.

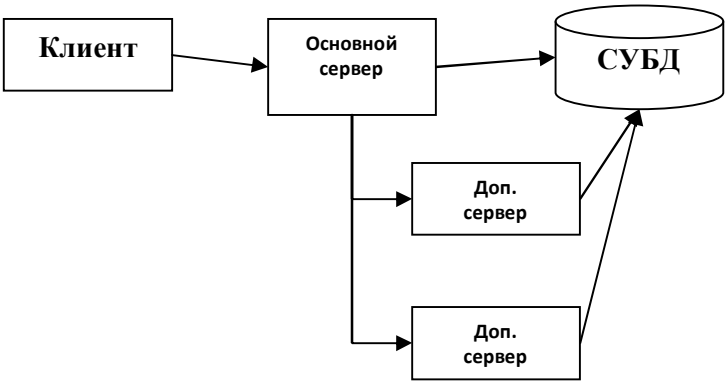


Рис. 2. Схема с мультиплексированием запросов (СМ)

Модифицированная схема основана на мультиплексировании входного потока запросов. В СМ (схема с мультиплексированием), как и в первоначальной схеме, основной прикладной сервер полностью обрабатывает быстрые запросы. Данные медленных запросов основной сервер передает на обработку дополнительным серверам и сохраняет уникальную идентификационную информацию для отправки ответа именно

тому клиенту, который инициировал запрос (например, имя пользователя в системе информационной безопасности). Количество дополнительных серверов может быть произвольным и подбирается в зависимости от конкретных условий эксплуатации. В общем случае дополнительные серверы могут быть разного типа, что определяется набором запросов, которые обрабатывает данный дополнительный сервер. Такой подход позволяет создавать регулируемую «полосу пропускания» для определенных запроса/запросов или отдельной операции.

При использовании схемы с мультиплексированием особое внимание следует обратить на взаимодействие серверов системы с реляционными СУБД. СМ требует параллельного обращения к базе данных, что при наличии конкурирующих sql – запросов, может привести к серьезному негативному влиянию блокировок СУБД: снижение производительности системы из-за сериализации sql – запросов; аварийное завершение sql – запроса и, следовательно, всей операции при возникновении взаимоблокировок. Методы исключения или минимизации влияния данных факторов заключаются, как правило, в учете особенностей выбранной СУБД и оптимизации бизнес – логики и ее реализации в прикладных серверах системы.

Если в ходе реализации проекта существует возможность выбора типа СУБД, то следует обратить внимание на системы с реализацией одновременного конкурентного доступа на основе многоверсионности (MVCC – MultiVersion Concurrency Control). К таким СУБД относятся, например, коммерческие продукты: Oracle (с версии 7), MS SQL (с версии 2005 Yukon); свободно распространяемые PostgreSQL, Firebird и т.д. Особенностью СУБД с использованием MVCC является то, что создание записи в таблице базы данных не блокирует операции просмотра, а просмотр информации не блокирует операции создания записей.

Проектирование, разработка и оптимизация логики прикладных серверов в части взаимодействия с СУБД в общем случае является неформальной задачей и каких-либо универсальных решений не существует. Однако на этом этапе существуют некоторые формальные технические аспекты, влияющие на эффективность взаимодействия с СУБД. Далее приводятся некоторые рекомендации разработчиков СУБД,

которые были проверены на практике авторами данной статьи. Следует отметить, что наибольший эффект от соблюдения данных рекомендации наблюдается в высоконагруженных транзакционных системах с высокой скоростью запросов к СУБД.

Связываемые переменные. При реализации компонент непосредственного доступа к СУБД, независимо от используемого интерфейса, крайне рекомендуется использование связываемых переменных при выполнении sql – запросов. Это позволяет СУБД использовать при повторном выполнении уже готовый план выполнения sql – запроса, что позволяет избежать многократного выполнения достаточно ресурсо – времяемкой операции подготовки плана выполнения. Нарушение данной рекомендации, например при непосредственной подстановке входных параметров запроса в текст sql – выражения, приводит к негативным последствиям независимо от типа используемой СУБД: повышенному времени отклика СУБД; повышенному использованию памяти, выделенной в СУБД, для хранения контекста запроса (план выполнения и т.д.) и т.д.

Однократная подготовка sql – выражения. Данная рекомендация заключается в том, что повторно используемые sql - выражения требуется подготавливать (отдельный запрос на уровне программного интерфейса СУБД с целью синтаксического/семантического разбора текста запроса и выработки плана выполнения) один раз без последующего удаления выработанного плана выполнения. Обоснование данной рекомендации дано в предыдущем пункте о связываемых переменных.

Блочные операции. В транзакционных системах львиную долю sql – запросов составляют операции создания/модификации данных в базе данных, или говоря терминами sql – INSERT/UPDATE. При пакетной обработке данных прикладным сервером, при которой в одном запросе сервер обрабатывает данные нескольких операций, требуется осуществить несколько запросов, связанных с созданием/модификацией записи в базе данных для каждой операции. При выполнении операции вставки/модификации осуществляется следующий технологический цикл взаимодействия с СУБД: передача данных в СУБД – выполнение запроса – передача результата выполнения в

программное обеспечение. Первый и третий этапы связаны с накладными расходами из-за протокольной упаковки/распаковки данных и передачи по каналам компьютерной сети. Временные задержки, необходимые на эти действия, ограничивают скорость выполнения sql – запросов и увеличивают время нахождения прикладного сервера в режиме ожидания ответа от СУБД, что снижает эффективность обработки данных. Решением данной проблемы является использование блочных операций, когда для одного выполнения sql – выражения возможно связывание целого массива входных параметров, определяющих действия (создание/обновление) над несколькими записями таблицы базы данных. Очевидно, что «удельные» накладные расходы при использовании блочных операций в определенных условиях меньше, чем при последовательно выполняемых «одиночных» запросов. На практике использование блочных операций в работе одного из прикладных серверов транзакционной системы с высоким потоком sql – запросов позволило в несколько раз снизить количество выполняемых к базе данных запросов и уменьшить время нахождения сервера в режиме ожидания ответа от СУБД в 1.5 – 2 раза.

Возвращаясь к схеме с мультиплексированием потоков, можно выделить следующие достоинства и недостатки схемы.

Недостатки:

- из-за параллельного взаимодействия с СУБД различных серверов есть риск возникновения взаимоблокировок. Для исключения завершения запросов по алгоритму разрешения взаимоблокировок, который реализован в СУБД, требуется тщательное проектирование проведения операций;
- в основном сервере требуется обязательное ведение очереди клиентских запросов. Медленные запросы помещаются в очередь при отсутствии свободных дополнительных серверов соответствующего типа с последующей обработкой по мере освобождения последних.

Достоинства:

- схема обеспечивает требуемое время отклика системы даже при наличии различных «медленных» запросов;
- при использовании данной схемы возможно организация «гарантированной полосы пропускания» для определенного типа

запросов (и не обязательно «медленных»), например, для проведения наиболее критичных к времени выполнения операций. При эксплуатации данная «полоса пропускания» может динамически увеличиваться путем запуска новых дополнительных серверов;

- при переходе с классического трехуровневого клиент-сервера не требуется модификация, как самих клиентских модулей, так и серверной подсистемы безопасности;
- параллельная обработка клиентских запросов повышает способность серверного программного обеспечения к вертикальному масштабированию;
- если основной и дополнительные серверы выполнены в виде отдельных процессов и используется соответствующее межпроцессное взаимодействие, то повышается способность серверного программного обеспечения к горизонтальному масштабированию;
- если основной и дополнительные серверы выполнены в виде отдельных процессов, то повышается отказоустойчивость системы. Так, выход из строя одного из дополнительных серверов (в случае, когда он не единственный) не приводит к отказу в обслуживании.

Схема с демультиплексированием

Схема с демультиплексированием (СДМ) имеет вид представленный на рис. 3.

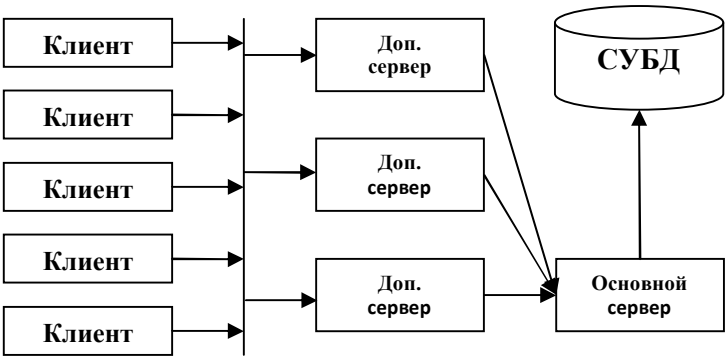


Рис. 3. Схема с демультиплексированием запросов (СДМ)

Как видно, схема расположения основного и дополнительных серверов обратная по отношению к схеме с мультиплексированием. Соответственно отличаются и условия применения, когда СДМ имеет преимущества перед другими схемами.

Один из возможных наборов условий применения СДМ: большое количество клиентских соединений (до нескольких тысяч) и соответственно большое количество запросов; кроме того, бизнес – логика системы позволяет дополнительным серверам обрабатывать часть запросов локально, т.е. без обращения к основному серверу (в противном случае, основной сервер становится «узким» местом и эффективность схемы резко падает). Последнее приведенное условие является наиболее критичным и может выступать в роли основного критерия применимости схемы в указанных условиях.

При реализации данной схемы на практике требуется решить вопрос, каким образом клиентский модули соединяются с дополнительными серверами. Возможные варианты:

- адрес дополнительного сервера для соединения хранится в локальной настройке клиентского модуля
- соединение осуществляется через дополнительное оборудование, обеспечивающее балансировку соединений клиентов с учетом всех работающих дополнительных серверов (в этом случае клиентский модуль настраивается на соединение с данным оборудованием)
- балансировка соединений реализуется одним из модулей системы (самостоятельная программная реализация эквивалента предыдущего варианта).

Все эти варианты имеют как свои достоинства, так и свои недостатки. Выбор оптимального варианта зависит от конкретных требований к системе.

Распределение клиентских соединений по целому пулу дополнительных серверов приводит еще к одной проблеме: идентификация/аутентификация любого клиента должна осуществляться в общем случае на любом из дополнительных серверов. Использование встроенных в дополнительные серверы штатных для Генератора проектов подсистем безопасности с локальным хранением идентифицирующей/аутентифицирующей

информации потребует периодического проведения достаточно трудоемкой операции по синхронизации баз данных пользователей и ключей (БДПК) различных серверов. В общем случае такой децентрализованный подход нельзя считать приемлемым для промышленной эксплуатации. С учетом такой ситуации был реализован централизованный подход к хранению БДПК и технологии идентификации/аутентификации пользователей. Для этого была разработана подсистема информационной безопасности на основе удаленного криптосервера (самостоятельный серверный модуль и инфраструктура обслуживания БДПК). В рассматриваемой схеме с демультиплексированием один удаленный криптосервер может обслуживать как дополнительный, так и основной серверы и обеспечивает централизованное хранение и управление данными БДПК.

Другая ситуация, когда может применяться СДМ, это реализация отдельных каналов взаимодействия различных внешних систем с основным сервером. В этом случае каждый дополнительный сервер (или некий выделенный пул серверов) служит интерфейсной точкой входа для внешней системы. Передача данных между точкой входа и внешней системой может осуществляться по протоколу, отличному от применяемого при внутрисистемном взаимодействии компонент системы. В этом случае дополнительный сервер реализует еще и функцию взаимного преобразования внутреннего и внешнего протоколов. При реализации следует обратить внимание на приоритет обработки основным сервером запросов от клиентских модулей различного типа. Так, в случае одновременной работы клиентских модулей типа рабочего места и дополнительных серверов (последние тоже являются клиентами по отношению к основному), в условиях одинакового приоритета обработки на соединение может наблюдаться некий «перекос» во времени обработки запросов. Это происходит из-за характера работы клиентских модулей разного назначения. Модули рабочего места, предназначенные для работы под управлением операторов системы, как правило, формируются одиночные синхронные запросы. Точки доступа, напротив, обычно работают в пакетном режиме, за один раз передавая несколько запросов. Это приводит к

тому, что либо логика работы основного сервера должна учитывать разные характеры формирования запросов от клиентских модулей разного назначения, либо необходимо принудительно сделать метод передачи запросов одинаковым для всех клиентских модулей. Второй метод «уравнивая» приоритетов можно реализовать, например, установкой дополнительного сервера между модулями операторов и основным сервером.

Подводя итоги по схеме можно выделить следующие характеристики схемы.

Недостатки:

- в общем случае достаточно серьезное усложнение системы за счет необходимости использования выделенного криптосервера и средств распределения запросов на соединение по пулу дополнительных серверов;
- необходимо либо в логике работы основного сервера, либо введением новых модулей системы учитывать разнотипность клиентских модулей (характер передачи данных запроса).

Достоинства:

- адаптация серверной части схемы для одновременного взаимодействия с большим количеством клиентских соединений;
- упрощение администрирования системы ввиду централизованного управления базой данных пользователей и ключей подсистемы информационной безопасности (особенно при большом количестве прикладных серверов различного назначения);
- при реализации дополнительных серверов в виде отдельных процессов, модульность серверной части схемы создает хорошие условия как к вертикальному, так и к горизонтальному масштабированию;
- запуск дублирующих процессов дополнительных серверов повышает общую отказоустойчивость системы.

ОКОННЫЙ ИНТЕРФЕЙС В ТЕХНОЛОГИИ ГЕНЕРАТОРА ПРОЕКТОВ

Н.И. Широков

Мировой опыт прошедших двух десятилетий массового применения оконного интерфейса в пользовательских программных системах заключается в том, что развитие программных средств происходило с сильным перекосом в технологичность. Было разработано огромное количество самых разнообразных библиотек и инструментальных систем, позволяющих писать программы с оконным интерфейсом. Постепенно сформировалась и стабилизировалась номенклатура элементов ввода и визуализации, таких как меню, кнопки, линейки прокрутки, акселераторы, поля ввода текста, таблицы, деревья и многие другие. Формировались и менялись стили использования элементов интерфейса, зачастую под влиянием модных поветрий и рекламных кампаний от крупных производителей программного обеспечения. Сейчас ми имеем массу возможностей писать программы с оконным интерфейсом.

На рынке программного обеспечения представлены десятки программных продуктов, которые позволяют разрабатывать программы с оконным интерфейсом любого стиля. Но при всем этом разнообразии языков, библиотек и инструментальных средств уровень разработки оконного интерфейса остается приблизительно таким, каким он был полтора-два десятилетия назад. Стала разнообразнее номенклатура кирпичиков и прочих строительных материалов, а технологии остались прежними. Объем программного кода в расчете на элемент интерфейса если и уменьшился, то совсем незначительно. Да, в языках Java или C# мы имеем более элегантные конструкции, нежели в Microsoft GUI Win32. Но все равно программист тонет в бесконечных деталях управления элементами интерфейса.

Практически отсутствуют инструментальные средства, которые бы обеспечивали высокую технологичность разработки оконного интерфейса. Без высокой технологичности стоимость разработки пользовательского оконного интерфейса составляет львиную долю стоимости всего проекта, особенно если идет речь о проектах различных учетно-аналитических систем и систем

управления. В условиях доминирования рыночных механизмов в индустрии программного обеспечения у разработчиков нет серьезной мотивации выставлять на рынок инструментальные системы такого типа. Объем рынка инструментальных средств на порядки меньше объема рынка программ для конечных пользователей. Потенциальный покупатель инструментальных средств по своей квалификации гораздо выше покупателя программ общего назначения. Из этого есть два следствия: во-первых, велик риск пиратского использования инструмента, во-вторых, грамотному разработчику зачастую требуется не сама инструментальная система, а опубликованная концепция, архитектура. Ознакомившись с таковой, разработчик самостоятельно применит в своих разработках опубликованные идеи. Ну и, наконец, крупные фирмы, работающие в программной индустрии, никоим образом не заинтересованы в продвижении на рынке инструментальных средств, потому что тем самым они только осуществляют технологическую поддержку своих потенциальных конкурентов. Крупная фирма за счет большой тиражности своей продукции или наличия дорогостоящих уникальных проектов располагает большими финансовыми возможностями, благодаря которым она не нуждается в высокотехнологичных инструментальных средствах. Проще использовать большое количество программистов невысокой квалификации, которые разрисуют программные интерфейсы и обеспечат их сопровождение.

Именно по этой причине наблюдается бесчисленное множество архаичных инструментов в чрезвычайно красивой упаковке. И бывает очень трудно объяснять заказчику, почему разработчик не может с легкостью снабдить заказанную им программу парой десятков разнообразных эффектных элементов интерфейса, увиденных в широко распространенных программах известных фирм. Цена, которую платит заказчик за экземпляр массового продукта крупной фирмы на порядки меньше платы за уникальную разработку, тираж которой – один или, если повезет, несколько экземпляров. А возможности такого разработчика, наоборот, на порядки меньше, чем у крупной фирмы. И без применения продуманных моделей интерфейса и технологий их реализации уникальные крупные проекты невозможны.

Эта статья посвящена описанию того, как в проблема построения оконного интерфейса решается в технологии Генератора проектов. Описание пользовательского интерфейса в проектной технологии разработки прикладных информационных систем является одной из трех главных составляющих проекта, наряду с описанием структуры данных и бизнес - процедур.

В первых версиях ГП модель интерфейса состояла из иерархической совокупности процедур, каждой из которых соответствовала некоторая экранная форма. В экранной форме в общем виде предусматривался ввод содержимого именованных полей, выполнение нескольких запросов к серверу и изображение результатов запросов в виде нескольких таблиц. Процедура имела совокупность входных параметров. Пользователь при созерцании результатов работы процедуры в экранной форме имел возможность выполнить выбранный пункт меню, предварительно выделив требуемые строки таблиц с данными. Результатом исполнения пункта меню могли быть: закрытие текущей процедуры с возвратом на предыдущую процедуру (вышестоящую в иерархии описания), выполнение запроса к серверу с обновлением содержимого экранной формы, вызов новой нижестоящей в иерархии процедуры поверх текущей. Интерфейс процедур был простой консольный: терминал VAX/VMS, потом экран MS DOS, далее эмулятор окна в MS Windows. Несмотря на достаточно примитивную модель интерфейса, она была использована для разработки нескольких реальных проектов в банковской сфере и не вызывала особых нареканий у конечных пользователей.

Однако затем в силу возросшей конкуренции и требований заказчиков (не путать с конечными пользователями) возникла настоятельная необходимость задействовать в генерируемых программах оконный интерфейс. Задача осложнялась тем, что как заказчики, так и разработчики, не могли сформулировать структуру этого интерфейса: хотелось чего-то “высокого и светлого”, но дальше пожеланий дело не шло. Здесь надо учесть, что используемая технология генерации программ предполагает обязательное наличие модели интерфейса. Традиционный подход с применением средств разработки программ с оконным интерфейсом не предполагает высокоуровневых моделей. Взамен

предлагается та или иная совокупность видов окон и многочисленные подпрограммы манипулирования данными в них. Такой подход хорош при разработке небольших по объему, но замысловатых по интерфейсу программ. Масштабы типовых проектов, разрабатываемых с помощью генератора, исключали использование стандартных средств, таких как MFC, Delphi, Gtk+, Qt и пр.

Первая попытка использования оконного интерфейса в генераторе основывалась на той же модели процедур, которая применялась в экранных формах. Типовое окно процедуры состояло из совокупности таблиц, теперь уже с линейками прокрутки, с настраиваемыми заголовками и выпадающими меню в дополнение к стандартному меню. Слева от таблиц располагалась древовидная структура, отражающая иерархию и хронологию вызовов процедур. Ввод данных в отличие от экранных форм выполнялся с помощью модальных диалогов с автоматически генерируемой разметкой с ограниченной возможностью влияния на нее.

После реализации оконного интерфейса и его обкатки на реальных проектах модель процедур была несколько модифицирована и дополнена с учетом особенностей нового интерфейса и отказом от экранных форм. Однако общая структура модели осталась прежней.

В начале 2000-х годов описанная модель исчерпала свой потенциал. Новые задачи в новых проектах требовали более эффективных решений. В связи с этим была разработана новая версия с принципиально новой моделью проекта. В этой версии приложение с оконным интерфейсом было полностью переделано. Модель приложения состоит из совокупности именованных видов окон. Каждое такое окно в общем виде состоит из главного меню окна, нескольких панелей инструментов, нескольких окон-панелей различного типа, строки состояния. Окна-панели в модели проекта описываются как отдельные именованные сущности, которые строятся из фиксированного в системе перечня оконных классов.

В распоряжении разработчика есть следующие оконные классы:

listview – таблицы с заголовками,

treeview – древовидные структуры,

textview – плоский (неформатированный) текст,
editview – простейший редактор плоского текста,
geom2dview – геометрические фигуры на плоскости,
geom3dview – геометрические фигуры в пространстве,
pictview – растровые картинки,
funcview – графики функций,
pageview – текст, организованный в таблицы,
hyperview – структурированный форматированный текст с встроенными объектами разного типа.

Отметим, что это не оконные классы из MS Windows GUI, а более высокоуровневые надстройки над ними или самостоятельно реализованные виды окон. Генерация программ приложений предполагает использование не только GUI для MS Windows, но и GTK+ для Unix-совместимых систем.

В основе описания оконного интерфейса лежит понятие команды. Команда – это именованная процедура базового языка, определенная в контексте либо приложения в целом, либо в конкретном окне верхнего уровня. В первом случае в программе доступен контекст окна приложения и приложения в целом (можно запускать несколько окон верхнего уровня). Во втором случае в процедуре доступен контекст конкретного окна, т.е. имеется возможность манипулирования окнами-панелями. Кроме уникального имени команда снабжена текстовым заголовком. Команда может быть указана в качестве выполняемого действия в главном меню окна, в контекстном меню окна-панели, в кнопке панели инструментов, в горячей клавиши окна верхнего уровня или окна-панели, реакции на перетаскивание файла в окно, начальное открытие окна, таймера и т.п.

Описание окна на основе встроенного в генератор оконного класса, по сути, мало отличается от использования аналогичных стандартных средств MFC, Delphi, GTK+ и т.п. Изобразительный уровень языка примерно такой же, только все в одном стиле базового языка и реализовано существенное подмножество свойств исходных оконных средств, имеющихся в операционных системах (требование независимости от конкретной системы). Такая модель по своей сути демонстрирует шаг назад от модели первой версии генератора, т.е. от непроцедурного описания интерфейса мы перешли к процедурному описанию.

Мотивация – значительно более гибкие средства конструирования интерфейса, ранее недоступные в первой версии. Это компромисс между краткостью описания первой версии генератора и практически неограниченными возможностями ручного трудоемкого программирования. В качестве альтернативы относительно низкоуровневого описания окон были разработаны макросредства описания типовых окон на основе конкретных специфицированных разработчиком документов. К таким типовым окнам относятся оконные типы wintable, wintree, partable. Табличное окно wintable – надстройка над оконным классом listview, в которой указывается набор структурированного документа, содержимое которого изображается в окне в виде таблицы. Реализован различные удобные средства управления содержимым окна (сортировка, выбор текущего элемента и пр.). Окно древовидной структуры wintree – надстройка над treeview. Древовидная структура строится на основе документа, типы узлов дерева задаются на специальном языке со ссылкой на наборы. Параметрическая таблица с вариантами partable – надстройка над listview. Изображает таблицу с колонками “Параметр”, “Значение”. Состав полей динамически вычисляется в зависимости от указанных предикатов.

Отдельный тип окон представляют диалоги – основные средства ввода информации. В отличие от первой версии ввод данных в новом оконном интерфейсе реализован не в составе текущего окна, а в модальном диалоге. Разработчик может в составе модели проекта описывать произвольное количество именованных диалогов. В описании диалога перечисляются элементы диалога – текстовые строки, поля ввода текста, поле выбора с выпадающим списком, радио-кнопки (с взаимоисключающим выбором), значки-картинки, кнопки исполнения, рамки, произвольные окна, описанные ранее. Порядок в списке определяет последовательность выбора по кнопке табуляции при работе с диалогом. Размещением перечисленных элементов можно управлять, задавая иерархическую структуру вертикальных и горизонтальных вложенных коробочек, содержащих элементы диалога. Размещение элементов производится на основе указанной информации автоматически. Можно (и нужно) задать реакцию на событие открытия диалога, в

котором можно заполнить элементы диалога конкретными данными, событие закрытия диалога по кнопке OK с извлечением данных из элементов диалога в программные переменные, а также реакцию на нажатие на кнопки диалога. Здесь также достигается компромисс между лаконичностью и жесткостью модели ввода в экранную форму первой версии генератора и неограниченными возможностями ручного рисования и программирования диалогов. Мотивация, как и в случае с окнами, – требование большей гибкости в конструировании интерфейса.

Есть одно макросредство в описании диалогов – dlgtable. Это короткое описание типового диалога, в котором производится выбор из нескольких таблиц, реализуемых макроокнами wintable. Для каждого описанного в модели проекта диалога в базовом языке доступны соответствующая структура и встроенная процедура. Вызов диалога заключается в заполнении полей структуры диалога, выполнению встроенной процедуры, проверки статуса закрытия диалога (OK/Cancel) и извлечении данных из структуры. Диалог может быть вызван из команды приложения и из события другого диалога. Модальность заключается в блокировании событий ввода вывода родительского окна/диалога, из которого вызывается данный диалог на время работ с ним.

Перечисленные выше оконные классы, из которых можно конструировать различные окна в приложениях, в основном ориентированы на использование в качестве интерфейса с пользователем, но не для построения результирующего документа и распечатки на принтере в виде твердой копии. Безусловно, существует простой метод формирования документа в каком-либо стандартном формате (rtf, html, xml и пр.), который впоследствии может быть просмотрен с помощью специальных программ, ориентированных на такой формат. Это не всегда бывает удобно, требовательные заказчики хотят иметь возможность работать с форматированными текстами и прочей графикой непосредственно в программе, которую они приобретают. Использование com-объектов для встраивания окон таких приложений, как MS Word, Acrobat Reader, Internet Explorer может решить эту проблему, но с известными ограничениями. Во-первых, если вы встраиваете некие средства от фирменного производителя в свою программу, то в этом случае заказчик должен будет устанавливать эти

продукты на каждый компьютер, где будет работать заказанная вам программа, зачастую именно той версии, на которую вы ориентировались. Во-вторых, сам по себе указанный интерфейс достаточно неудобен, и интеграция сложнейшего программного обеспечения в свою программу чревата существенным снижением надежности системы в целом. И, в-третьих, что самое главное, эта технология не будет работать на отличных от MS Windows перспективных платформах. А на этих перспективных платформах похожие технологии не развиваются по первым двум причинам. В связи с этим возникла потребность в разработке средств визуализации сложных объектов достаточно произвольной структуры, включающей в себя и форматированный текст, и другие графические объекты. При этом необходимо обеспечить перенос этого объекта с экрана монитора на бумагу эквивалентным образом, т.е. то, что пользователь видит на экране, он должен видеть и в напечатанном документе. Для решения указанной задачи в генераторе был разработан специальный оконный класс hyperview. Для представления графических объектов в hyperview, в состав генератора включен системный пакет, в котором представлен структурированный документ hyp. Схема документа приведена на рис. 1.

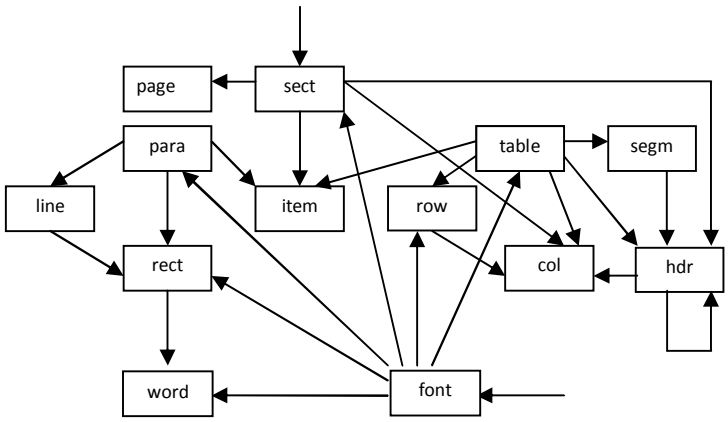


Рис. 1. Схема документа hyp для класса hyperview

На рис. 1 прямоугольники с надписями обозначают типы записей структурированного документа, стрелки – отношения между записями типа один ко многим. Имена записей указаны в надписях в прямоугольниках. Имена наборов на рисунке опущены, в тексте наборы будут идентифицироваться по паре (владелец набора, член набора), стрелка от владельца к члену. Стрелка без записи владельца – сингулярный набор, представлен одним экземпляром в документе (владелец – документ в целом).

Объект, изображаемый в окне, на верхнем уровне является упорядоченной последовательностью секций (запись sect), последовательность формируется сингулярным набором. Секция - это последовательность элементов (запись item, набор sect --> item). Элемент может быть двух типов: параграф (запись para) или таблица (запись table). Для привязки параграфа или таблицы к элементу используются соответствующие наборы (para --> item, table --> item), в которых отношение один ко многим вырождается в отношение один к одному. Набор используется как ссылка от члена набора к владельцу.

Параграф задается как последовательность прямоугольников (запись rect). Прямоугольник в данной модели является родовым понятием и может быть различного типа (расширяемого по мере развития модели). В простейшем и основном случае это неразрываемый текст, представленный последовательностью слов (запись word). Разделение неразрываемого текста на последовательность слов нужно для того, чтобы разные части текста (слова) могли снабжаться разными характеристиками – атрибутами, в данном случае это шрифт (запись font), размер и стиль шрифта, цвет и т.п.

Запись font организована в виде список посредством сингулярного набора. На шрифт предусмотрены ссылки от разных сущностей модели. Это сделано для удобства формирования иерархического объекта. Если для нижестоящего элемента не задан шрифт (или другой атрибут), то берется значение от вышестоящего элемента.

Запись line используется для представления результатов макетирования изображаемого объекта. Макетирование - это размещение прямоугольников модели по строкам слева направо до заполнения строки и далее сверху вниз. Размер каждого

прямоугольника вычисляется в зависимости от его типа. В рассмотренном случае текста этот размер вычисляется как функция используемого шрифта и его атрибутов. Для других типов прямоугольников применяется другой алгоритм вычисления. В результате макетирования создаются экземпляры записей line и устанавливаются связи по соответствующим наборам. Для каждой строки вычисляется ее размеры и координаты относительно параграфа. Предполагается, что параграф начинается с новой строки. При макетировании учитываются масса нюансов, задаваемых атрибутами секции, параграфа, прямоугольника, слова. В частности предусмотрено выравнивание влево, вправо, по центру, по ширине.

Другой тип элемента - это таблица (запись table). Таблица имеет заголовок - последовательность записей (hdr), организованных в иерархический список (наборы table --> hdr, hdr --> hdr). Содержимое каждого элемента заголовка является секцией (ссылка по набору sect --> hdr). Таким образом объект, изображаемый в рассматриваемом оконном классе, имеет рекурсивную структуру. В простейшем случае элемент заголовка будет секцией (sect) из одного элемента (item) – параграфа (para) из нескольких прямоугольников (rect) слов. В общем случае элемент заголовка может быть представлен секцией, состоящей из параграфов и других таблиц и т.д. вглубь иерархии (если кому-то такое чудо понадобится). Каждая запись типа hdr имеет заданную относительную ширину. Абсолютная ширина вычисляется сверху вниз по иерархии заголовков пропорционально значениям относительных значений ширины в каждом уровне. Если секция пуста, то ее высота в результате макетирования получается нулевой, и она выпадает из визуального представления объекта.

Следующий компонент таблицы – список ее строк (запись row, набор table --> row). Каждая строка таблицы представлена списком ячеек строки (запись col, набор row --> col). Ячейка строки имеет ссылку на элемент заголовка нижнего в иерархии уровня (набор hdr --> col). Содержимое ячейки представляет собой произвольную секцию (ссылка на нее по набору sect --> col). Здесь тоже проявляется рекурсивность в определении объекта. Так, ячейка таблицы может быть произвольной секцией, т.е. состоять из других таблиц.

Третий тип элемента (item) секции - это растровое изображение. При формировании объекта программист может встроить в модель произвольное количество растровых картинок и в данном конкретном элементе сослаться на любую из них по индексу в пределах структурного документа. Для растровой картинки задается масштаб – соотношение пикселей растра (не устройства, монитора или принтера) с метрическими единицами. Дело в том, что вся геометрия изображаемого объекта при макетировании строится не в пикселях, а в метрических единицах (twip – 1/1440 дюйма). Пиксели зависят от устройства, на котором изображается объект, экран монитора или принтер, при этом размер пикселя в twips на разных устройствах разный.

Существует несколько способов формирования объектов для класса hyperview. Первый способ построения объекта для hyperview - это написание программы в составе приложения (или окна, входящего в приложение), которая запись за записью, набор за набором сформирует описанную выше структуру. Такой способ малопригоден практически, так как требует от разработчика знаний о правилах и ограничениях, которых надо придерживаться при назначении многочисленных атрибутов записей модели.

Второй способ, это построение модели не вызовом встроенных процедур манипулирования записями и наборами структурированного документа hyp, а вызовом специально для этого случая написанных процедур в составе системного пакета hyp. Эти процедуры выполняют похожие действия, но корректно и согласованно устанавливают атрибуты создаваемых записей и производят дополнительный контроль целостности, не позволяя создавать некорректные структуры. Такой способ интуитивно понятен при наличии минимальной инструкции и/или представительного примера программы.

Третий способ, это использование специального языка, на котором в текстовом файле с интуитивно понятным синтаксисом описывается структура объекта. В составе системных пакетов генератора поставляется программа, которая производит чтение такого файла, синтаксический анализ и формирование описанной модели для визуализации в оконном классе.

ОРГАНИЗАЦИЯ ФУНКЦИОНИРОВАНИЯ БИЛЛИНГОВОЙ СИСТЕМЫ В АВТОМАТИЗИРОВАННОЙ БАНКОВСКОЙ СРЕДЕ

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

В статье рассматривается круг задач, связанный с проектированием и разработкой биллинговой системы в среде крупного коммерческого банка.

Эффективное развитие по-настоящему массового розничного обслуживания клиентов в коммерческих банках требует внедрения новых технологий. Одной из таких технологий является биллинг. Биллинговые системы, по сути, являются посредниками между населением, обслуживающими предприятиями и банками. Применение биллинга позволяет упростить и автоматизировать процедуры приема и обработки платежей населения путем предварительного сбора информации о задолженностях клиентов и актуализации ее в момент совершения платежа. С помощью биллинга банки автоматизируют важный сектор своей работы - платежи населения, избавляются от ручного ввода платежных 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 приема платежей с помощью банкоматов, инфокиосков, электронных карт и прочих устройств самообслуживания.

Автоматизированная система биллинга - это информационно-учетная система, ориентированная на обеспечение взаиморасчетов между поставщиками и потребителями услуг. Поставщики услуг выставляют счета на оплату предоставленных 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.

Автоматизированная система биллинга, о которой пойдет здесь речь, обеспечивает:

- возможность клиентам банка в режиме on-line (24x7) выполнять необходимый набор операций по любому доступному каналу;
- передачу информации о выполненных платежах в АБС банка;
- передачу информации о выполненных платежах поставщикам услуг;
- необходимый уровень информационной безопасности;
- ведение детального учета клиентов банка, включая их индивидуальные технологические особенности обслуживания (в виде персональных профилей);
- ведение реестра задолженностей зарегистрированных клиентов,
- ведение справочника поставщиков услуг;
- ведение перечня услуг, по которым принимаются платежи;
- информационное обслуживание клиентов;
- обслуживание различных каналов взаимодействия с клиентом;
- гибкую настройку импорта/экспорта данных в режиме off-line;
- для целей поддержки оперативной деятельности клиентов взаимодействие в различных режимах с широким спектром внешних автоматизированных систем;
- разделение системы на транзакционную и аналитическую составляющие;
- обеспечение масштабируемости транзакционной составляющей в части скорости выполнения клиентских операций;

Система реализована по технологии Генератора проектов, в основе которой лежит проектный подход к созданию прикладных информационных систем. Базовой архитектурой системы является трехуровневая клиент-серверная схема. Хранение данных в системе реализовано средствами реляционной СУБД. Основными информационными объектами БД являются справочники, в которых хранятся и модифицируются все содержательные данные системы.

Справочники системы

Основные справочники, задействованные в биллинговой системе:

- справочник договоров;
- справочник операций;
- справочник дополнительных реквизитов операций;
- справочник поставщиков услуг;
- справочник лицевых счетов и задолженностей;
- справочник клиентов и их профилей;
- справочник параметров комиссионных сборов;
- справочник платежных средств;
- справочник платежных отделений банка;
- справочник устройств самообслуживания;
- справочник банков;
- справочник аналитических счетов;
- справочник совершенных операций;
- справочник доп. реквизитов совершенных операций.

Справочник договоров содержит информацию о заключенных банком договорах с поставщиками услуг. Информация о договоре включает в себя сроки начала и окончания действия, уровень привязки – отделение банка или весь банк целиком.

Справочник операций содержит информацию о технических параметрах выполнения операции в биллинговой системе, включая в себя многочисленные признаки и опции, такие как:

- онлайн операции;
- офлайн операции;
- операции по лицевым счетам поставщиков услуг;
- операции по банковским счетам платежных средств.

Справочник дополнительных реквизитов операции предназначен для дополнительной информационной поддержки операции.

Справочник поставщиков услуг содержит информацию о наименовании поставщика услуг, его банковских реквизитах и дополнительных признаках и опциях.

Справочник лицевых счетов и задолженностей содержит информацию о владельце лицевого счета, его контактах и параметрах задолженности. Справочник клиентов и их профилей содержит удостоверяющую личность информацию о клиентах банка, а также информационную структуру о персональных наборах параметризованных операций, привязанных к конкретному клиенту. Справочник параметров комиссионных сборов содержит информацию о правилах взимания комиссии с проводимой операции. Справочник платежных средств содержит информацию о платежных картах клиентов, зарегистрированных в банке. Справочник платежных отделений банка содержит информацию о структуре отделений банка, банковских реквизитах отделений. Справочник устройств самообслуживания определяет доступные в банке устройства для выполнения операций. Справочник банков (поставляется из ЦБ РФ) используется для контроля банковских реквизитов при ручном вводе операций. Справочник аналитических счетов используется для формирования документов при выполнении операций выгрузки. Справочник совершенных операций содержит информацию о совершенных операциях клиентов. Справочник доп. реквизитов совершенных операций дополняет справочник совершенных операций.

Модель проведения операций

Под проведением операции в биллинговой системе подразумевается последовательность действий в бизнес-логике транзакционного сервера, инициированная клиентом или автоматически по расписанию системы. Эта последовательность формирует изменения в справочниках системы.

Самая массовая операция в биллинговой системе – это оплата услуг. Типовое проведение операции сводится к следующим действиям:

- поиск поставщика услуги по реквизитам;
- ввод идентифицирующих реквизитов лицевого счета;
- ввод основных и дополнительных реквизитов лицевого счета;
- контроль введенных реквизитов на сервере системы;
- регистрация совершенного платежа в справочнике.

Если клиент использует оплату услуг через личный профиль, то поиск поставщика услуг сводится к выбору необходимой операции в профиле (например, «Платеж за свой телефон»), а ввод всех реквизитов происходит автоматически на сервере из профиля клиента, т.е. клиенту необходимо указать только сумму, а если происходит оплата задолженности, то сумма подставляется автоматически.

Алгоритмы проведения операций в системе задаются как в настройках операции, так и в специализированных скриптах, которые позволяют достаточно гибко изменять сценарий обработки операции.

Архитектура биллинговой системы

Биллинговая система реализована в классической архитектуре трехуровневого клиент-сервера. Такая архитектура автоматически формируется Генератором проектов при описании клиент-серверной модели бизнес - процессов. Модель бизнес - процессов задается в виде описания перечня и структуры запросов от клиентский приложений к бизнес-серверам системы, а также в виде декларации содержательной части бизнес-запросов и запросов к базам данных. Обеспечение сетевого взаимодействия и информационной безопасности реализовано на уровне автоматически генерируемого программного кода.

С точки зрения прикладной архитектуры разделение системы на транзакционную и аналитическую составляющие необходимо для обеспечения независимого выполнения быстрых и медленных операций в системе. В этих целях транзакционный и аналитический функционал разнесен по разным серверам системы.

В транзакционной системе часто возникает ситуация, в которой через один и тот же канал требуется выполнение достаточно медленных запросов на фоне большого количества очень быстрых и коротких запросов. Чтобы медленные запросы не влияли на время отклика быстрых запросов, необходимо реализовать схему, в которой сервер, обслуживающий канал поступления всех запросов, обрабатывает быстрые запросы сам, а медленные запросы передает на выполнение специальным дополнительным серверам, являющимися клиентами к основному серверу. При необходимости в случае занятости всех

дополнительных серверов основной сервер должен ставить медленные запросы в очередь и по мере освобождения дополнительных серверов, должен передавать запросы из очереди на исполнение. Такая схема работы позволяет распараллелить запросы при соответствующей поддержке СУБД и правильном распределении типов запросов по серверам.

На рис.1 показана архитектура биллинговой системы.

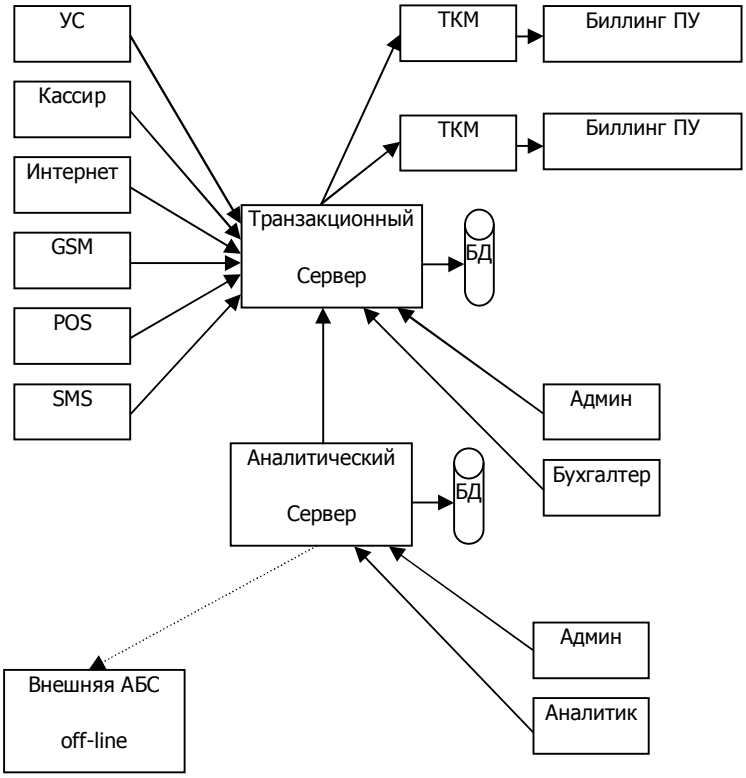


Рис. 1. Многоканальная биллинговая система

Многоканальность

Многообразие каналов обслуживания позволяет доставлять различные банковские услуги клиенту в наиболее удобной и доступной форме.

Биллинговая система поддерживает большую номенклатуру каналов обслуживания:

- сети устройств самообслуживания (банкоматов, инфокиосков);
- модули кассиров-операторов в отделениях банка;
- интернет (выполнение операций через браузер);
- GSM (мобильный телефон с установленным на него приложением);
- сети POS-терминалов;
- SMS (отправка форматированных SMS через мобильный телефон).

Все перечисленные каналы функционируют в режиме on-line в едином информационном пространстве системы (клиенты, поставщики услуг, лицевые счета, задолженности, документы). Это означает, что клиент может работать со своим профилем, со своими лицевыми счетами единообразно через все каналы обслуживания.

Канал устройств самообслуживания исторически появился первым в биллинговой системе, что наложило свой отпечаток на архитектуре системы, отразившееся на работе с различными реквизитами операций, особенно текстовыми, вводимыми клиентами на устройствах самообслуживания. Но со временем появление новых каналов позволило абстрагироваться от особенностей урезанного интерфейса устройств самообслуживания и реализовать полноценные интерфейсы с пользователем на остальных каналах биллинговой системы.

Наиболее востребованным каналом обслуживания биллинговой системы является канал Кассиров-операторов. Это объясняется тем, что большинство клиентов по привычке приходят в отделения банка для оплаты услуг, предпочитая живое общение вместо технического устройства из-за опасения «сделать что-нибудь не так».

Канал обслуживания интернет наиболее перспективный в силу того, что сеть интернет развивается стремительными темпами, а ее технические возможности по передаче данных и интерфейсу с пользователем превосходят все остальные. К тому же, в этом канале нет живых очередей, что психологически для

многих клиентов является огромным преимуществом, позволяя неспешно разобраться с выполнением необходимых операций. Плюс к этому доступность канала круглосуточно и в любой день недели обеспечивает клиентам оперативность выполнения операций.

Еще один канал обслуживания биллинговой системы – GSM-канал – удобен тем, что всегда «под рукой». Мобильные телефоны получили широчайшее распространение, и среди них велика доля тех, на которые есть возможность установить мобильное приложение с поддержкой биллинговых услуг. Безусловно, интерфейс мобильных телефон заметно скуднее, нежели интернет браузеров, но тем не менее возможностей для проведения большинства операций, особенно сохраненных в профиле клиента, хватает с лихвой. А учитывая стремительное развитие и распространение смартфонов и коммуникаторов, грань между настольными компьютерами и мобильными телефонами тускнеет с каждым годом.

Наконец, сеть POS-терминалов - самый необычный канал обслуживания биллинговой системы - представляет собой специальное ПО, встраиваемое в прошивку POS-терминалов, которое взаимодействует с серверами биллинговой системы. Наиболее востребован такой канал на торговых точках, оборудованных POS-терминалами для оплаты услуг, которые при этом позволяют дополнительно проводить операции клиентов в биллинговой системе.

Что касается каналов вообще, то стоит упомянуть, что биллинговая система открыта для любых новых каналов, укладывающихся в модель проведения операций.

Масштабируемость

В биллинговой системе с ростом количества выполняемых операций возрастает нагрузка на транзакционную составляющую системы. В какой-то момент интенсивность поступления запросов может превысить производительность системы и запросы на выполнение операций начнут выстраиваться в очередь, что неизбежно приведет к тайм-аутам обработки. Для исключения подобных ситуаций в биллинговой системе реализованы

различные механизмы, позволяющие достичь приемлемой масштабируемости.

В частности, реализованы дополнительные сервера, позволяющие вынести выполнение тяжелых и продолжительных запросов в фоновое исполнение, не мешая выполнению легких и быстрых запросов.

Для равномерного и линейного распределения нагрузки по каналу кассиров-операторов реализованы специальные фронтальные сервера, обеспечивающие поддержку множественных соединений с клиентскими модулями, передавая запросы от них через мультиплексное соединение.

Взаимодействие с внешними автоматизированными системами в режиме off-line

Первоначально биллинговая система разрабатывалась для обеспечения онлайнового доступа к офлайновой АБС. Это объяснялось тем, что большинство банковских АБС того времени не обладали возможностью работы в онлайн режиме. Для решения этой проблемы был реализован механизм псевдоонлайн, т.е. взаимодействие с АБС осуществляется через файловый обмен, а с клиентами системы через онлайн.

Механизмы файлового обмена реализованы в максимально гибкой форме с использованием многочисленных параметрических выборок из базы данных и с дополнительным применением скриптов импорта/экспорта для формирования и обработки многочисленных форматов файлов. Интерпретатор скриптового языка формируется Генератором проектов на основании описания проекта и интегрируется в серверную компоненту системы.

Процедур импорта/экспорта могут быть инициированы как по запросу оператора банка в ручном режиме, так и по расписанию в автоматическом режиме.

Использование расписаний позволяет настроить импорт/экспорт таким образом, чтобы ресурсоемкие процедуры выполнялись в наименее загруженное время работы системы.

Наиболее востребованные процедуры импорта — это импорт лицевых счетов поставщиков услуг и импорт справочника платежных средств.

Наиболее востребованные процедуры экспорта — это экспорт совершенных операций.

Взаимодействие с внешними системами в режиме on-line

Для обеспечения оперативного отражения операций клиентов в биллинговой системе реализован механизм on-line взаимодействия с внешними автоматизированными системами. Для этих целей для поставщика услуг, поддерживающего on-line взаимодействие, настраивается специальный транспортный клиентский модуль, который обеспечивает on-line связь с сервером поставщика услуг. Через этот модуль транслируются все запросы для этого поставщика услуг, позволяя в режиме реального времени изменять состояния лицевых счетов в конечном биллинге.

Протоколы и механизмы взаимодействия задаются в специализированных библиотеках, интерфейс с которыми со стороны транспортного клиентского модуля стандартизован, а со стороны внешней автоматизированной системы определяется ее спецификациями.

Часть библиотек разработана на языке Генератора проектов, а часть - реализуется с применением специализированного расширения Генератора проектов — так называемых Си-пакетов, которые позволяют реализовать сложные механизмы взаимодействия на языке Си.

Информационная безопасность

Защита обмена данными в режиме off-line (импорт/экспорт) осуществляется применением ЭЦП. В случае ручного экспорта формируется персональная ЭЦП оператора, в случае автоматического — ЭЦП автоматизированной системы.

Взаимодействие с внешними автоматизированными системами также может быть реализовано в виде онлайновых сетевых соединений с применением соответствующего контура информационной безопасности, который обеспечивается стандартными средствами сетевого взаимодействия, формируемого Генератором проектов.

АВТОМАТИЗИРОВАННАЯ СИСТЕМА ВЕСОВЫХ РАСЧЕТОВ В САПР ЛА

Н.И. Широков

Автоматизированная система весовых расчетов (АСВР) является одной из подсистем САПР ЛА. АСВР предназначена для обеспечения решения следующих важнейших задач весового проектирования летальных аппаратов:

- построение весовой модели изделия,
- задачи весового контроля и весового анализа,
- задач геометрии масс.

История разработки АСВР берет свое начало в 70-х гг. Первая версия была реализована на БЭСМ-6 силами сотрудников ВЦ РАН и МФТИ в рамках создания первой очереди САПР в ОКБ им. П. О. Сухого [1]. Тогда в связи с внедрением АСВР в конструкторском бюро были проведены организационные мероприятия, предусматривающие при изготовлении любого чертежа заполнение весовой формы для деталей и агрегатов, изображенных на чертеже. Весовая форма была разработана специально для АСВР. Без весовой формы чертеж не принимался. В первых версиях системы информация с весовых форм набивалась на перфокартах, вводилась в ЭВМ для проведения расчетов. С течением времени система модифицировалась, перфокарты вместе с БЭСМ-6 ушли в прошлое, появилась солидная VAX/VMS с интерактивным вводом, но технология весового проектирования с применением старой АСВР сохранялась вплоть до недавнего времени. В начале 2000-х гг. появилась настоятельная необходимость существенно модернизировать систему (ушла в прошлое и VAX/VMS). В настоящей статье представлена новая версия АСВР, разработанная с помощью Генератора проектов. Технология применения Генератора проектов состоит в разработке проекта создаваемой прикладной системы на языке Генератора, автоматической генерации по разработанному проекту программного комплекса на языке С и сборке исполняемых программ [2].

Основными программными компонентами системы являются сервер системы, рабочее место администратора безопасности и рабочее место пользователей, решающих задачи

весового проектирования. Рабочие места взаимодействуют с сервером по протоколу TCP/IP. Сервер обеспечивает взаимодействие администратора с базой данных (справочником) пользователей и взаимодействие пользователей – «весовиков» с базой данных весовых моделей изделий.

Весовая модель изделия

База данных весовых моделей является центральным информационным объектом системы. Весовая модель изделия представляет собой древовидную структуру конструктивного членения изделия. Корень дерева – изделие в целом, нижестоящие узлы – отдельные агрегаты, терминальные вершины – отдельные детали, из которых собираются агрегаты, узлы и в конечном счете изделие в целом. Терминальные (далее по принятой в системе терминологии висячие) вершины дерева содержат такие параметры, как массу, координаты центра масс и значения осевых, плоскостных, центробежных моментов инерции.

Весовая модель изделия по мере проработки проекта изделия строится сверху вниз от корня дерева – изделия к деталям. На ранних этапах проектирования в качестве деталей могут выступать достаточно крупные агрегаты. Массово-инерционные характеристики таких агрегатов на ранних этапах проектирования задаются методом экспертных оценок, по прототипам, упрощенным формулам или из каких-нибудь других соображений. По мере развития проекта вершины весовой модели детализируются. В этом случае массово-инерционные характеристики агрегатов вычисляются уже суммированием по нижестоящим вершинам. На более поздних стадиях проектирования характеристики деталей изделия вычисляются и вводятся на основании конструктивных схем и чертежей. В зависимости от способа вычисления весовых характеристик в весовой модели на разных этапах проектирования используются различные типы масс: лимитная, теоретическая, чертежная, фактическая, текущая. Правила манипулирования этими типами характеристик отражают сложившуюся практику работы в проектном подразделении.

Для удобства задания координат центра масс в системе предусмотрено ведение еще одной иерархической структуры -

дерева применяемых систем координат. Каждая система координат снабжается уникальным наименованием и параметрами привязки к вышестоящей системе (три координаты смещения начала координат и три угла поворота в определенном порядке вокруг координатных осей). Каждая вершина весовой модели ссылается на некоторую систему координат, в которой заданы ее координаты центра масс.

Вершины весовой модели снабжены несколькими классификационными параметрами (материал, отдел, …) для целей группировки массово-инерционных характеристик по принадлежности различным классификаторам. Группировка вершин дерева конструкции может проводиться также по самолетным подсистемам (гидравлика, топливная подсистема, электрооборудование и прочие), перечень которых вводится в весовую модель. В результирующих весовых сводках можно осуществлять выдачу информации как по конструктивному членению, так и по классификационным признакам или по принадлежности к разным подсистемам.

Для учета массово-инерционных характеристик заклепок, болтов и гаек, окраски и т.п. реализовано такое понятие, как «крепеж». В системе определены правила работы с «крепежом» и распределение «крепежа» по вершинам весовой модели.

Для построения динамически подобных геометрических моделей, применяемых в прочностных расчетах, изделие разбивается на пространственные отсеки, заданные выпуклыми многогранниками. В весовой модели предусмотрено задание таких отсеков с помощью набора плоскостей или, когда этого достаточно, параллелепипедами. Вычисление массово-инерционных характеристик отсека заключается в переборе всех висячих вершин модели с проверкой на принадлежность ее к данному отсеку. После распределения вершин по отсекам производится суммирование массово-инерционных характеристик всех входящих в отсек вершин.

В системе предусмотрено задание так называемых сосредоточенных масс, когда некоторые не висячие вершины весовой модели объявляются принадлежащими к поименованной сосредоточенной массе (аналогично подсистемам). При вычислении массово-инерционных характеристик отсеков висячие

вершины в составе сосредоточенных масс не подлежат распределению по отсекам, выделяются отдельной строкой как единое целое.

Изделия, описываемые весовой моделью характеризуются явно выраженной симметрией (левая и правая консоли крыла и оперения, симметрично расположенные мотогондолы, двигатели, стойки шасси и тому подобное). Весовая модель в целях экономии предусматривает возможность задавать признаки симметрии у отдельных агрегатов и этим сокращать объем хранимой информации и время расчетов.

База данных весовой модели

В базе данных могут храниться несколько независимых весовых моделей. Весовые модели хранятся в базе данных сетевой структуры под управлением системы GENBD, которая является компонентой Генератора проектов. У многих возникает вопрос: почему не используется одна из коммерческих баз данных вроде Oracle, MS SQL Server, или свободных систем типа MySQL, PostgreSQL, или файловой базы SQLite?

Основная причина выбора сетевой базы данных, а не стандартной реляционной является скорость работы. Весовая модель характеризуется наличием большого количества связей между записями, традиционно реализуемыми в реляционных системах связками “первичный/внешний ключ”. В перечисленных реляционных системах такие ограничения целостности автоматически проверяются в процессе модификации данных поиском по индексам. Несмотря на впечатляющие достижения в этой области время поиска по ключу пропорционально логарифму количества записей в полной таблице. В сетевой базе такие связи реализуются прямыми физическими ссылками, доступ по которым не зависит от полного размера таблицы. При сложных расчетах, когда имеется несколько вложенных циклов обхода таблиц, указанное свойство сетевой базы может стать решающим в вопросе эффективности.

Иерархическая структура данных (весовая модель представлена древовидной структурой систем координат и конструкций) затрудняет обработку данных посредством умело написанного сложного запроса. Приходится организовывать

рекурсивные процедуры и программные циклы обхода деревьев, причем на каждом шаге необходимо делать отдельный запрос к базе для получения перечня непосредственных потомков текущей вершины дерева. Рекурсивный обход дерева конструкций для вычисления суммарных характеристик агрегатов сопровождается дополнительными обходами поддеревьев, для которых необходимо перед суммированием распределить параметры крепежа или поправки на фактическую массу. И все это повторяется столько раз, сколько отсеков участвует в расчете. Поэтому доступ к данным в системе характеризуется большим количеством мелких запросов. Такой режим как раз эффективен в сетевых базах данных в отличие от реляционных, где эффективнее небольшое количество максимально сложных запросов в надежде, что "умная" система управления найдет максимально быстрый сценарий поиска. На рис. 1 приведена схема базы данных для хранения весовых моделей.

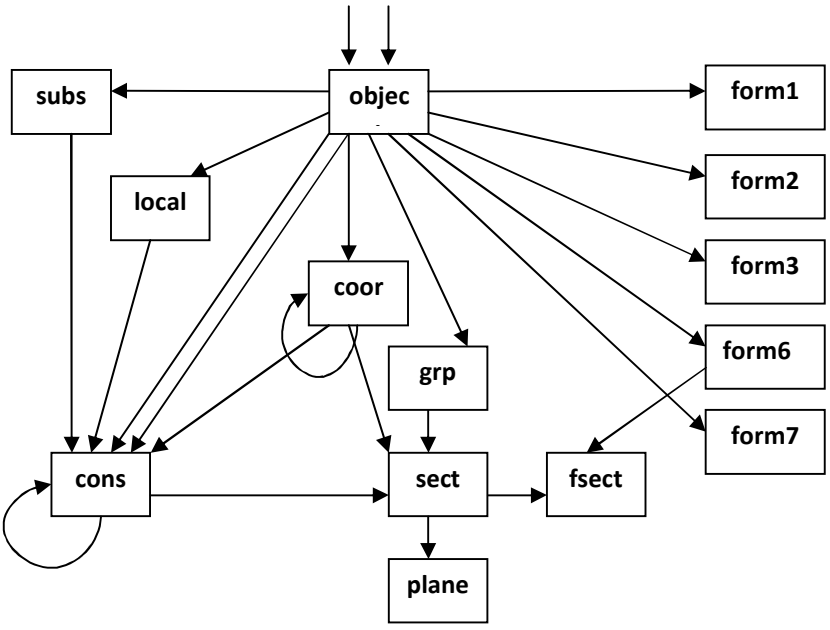


Рис. 1. Схема базы данных весовой модели

На приведенной схеме используется традиционная нотация сетевых баз данных. Прямоугольники обозначают типы записей (таблицы), стрелки обозначают наборы – отношения “один ко многим” или сингулярные наборы (висячие стрелки). Ниже приводятся списки типов записей и наборов весовой модели.

ЗАПИСИ:

- **object** – изделие. Содержит наименование. К изделию привязаны остальные компоненты модели.
- **cons** – конструкция (деталь, узел, агрегат). В составе атрибутов присутствуют: наименование, разного рода массы, массово-инерционные характеристики, различные признаки и пр.
- **coor** – система координат. Содержит наименование, положение начала координат относительно вышестоящей системы, углы поворота вокруг осей, а так же вычисляемую матрицу преобразования координат.
- **subs** – подсистема. Применяется для привязки конструкций к подсистемам изделия независимо от принадлежности к дереву конструкции.
- **local** – сосредоточенная масса. Применяется для группировки конструкций для предотвращения распределения их терминальных компонент по отсекам.
- **grp** – группа отсеков. Отсеки организованы в группы для удобства формирования выходных форм.
- **sect** – пространственный отсек. Содержит наименование, параметры ограничивающего прямоугольника.
- **plane** – ограничивающая плоскость отсека. Содержит тип задания плоскости: коэффициенты уравнения плоскости или координаты трех точек плоскости в заданном порядке и собственно указанные коэффициенты и координаты.
- **form1, form2, form3, form6, form7** – задания на расчет соответствующих выходных форм. Содержат параметры расчетов, наименования форм, конструкций, систем координат, в которых надо выдавать информацию, типы масс, признаков и пр.
- **fsect** – ссылочная запись для формирования перечня отсеков, участвующих в конкретном расчете формы 3.

НАБОРЫ:

- **amn-->object** – сингулярный набор, задающий перечень весовых моделей различных изделий в базе данных.

- **Trash-->object** – сингулярный набор, задающий перечень весовых моделей изделий в корзине, т.е. удаленных, но с возможностью восстановления.
- **object-->coor** – привязка систем координат к изделию.
- **object-->cons** – привязка конструкций к изделию.
- **trash-->cons** – привязка конструкций к корзине, т.е. удаленных, но с возможностью восстановления.
- **object-->subs** – привязка подсистем к изделию.
- **object-->local** – привязка сосредоточенных масс к изделию.
- **object-->grp** – привязка групп отсеков к изделию.
- **object-->form1, object-->form2, object-->form3, object-->form6, object-->form7** – привязка заданий на расчет к изделию.
- **coor-->coor** – привязка систем координат к вышестоящим системам. Организует иерархию систем, т.е. древовидную структуру.
- **cons-->cons** – привязка конструкций к вышестоящим конструкциям. Организует иерархию конструкций, т.е. древовидную структуру.
- **subs-->cons** – привязка конструкций к подсистемам. В дополнение к древовидной структуре некоторые конструкции могут принадлежать к одной подсистеме.
- **local-->cons** – привязка конструкций к сосредоточенным массам. В дополнение к древовидной структуре некоторые конструкции могут принадлежать к одной сосредоточенной массе.
- **grp-->sect** – привязка отсека к группе.
- **sect-->plane** – привязка плоскости к отсеку.
- **coor-->cons** – привязка конструкции к системе координат. Означает, что координаты ее центра масс заданы в данной системе координат.
- **coor-->sect** – привязка отсека к системе координат. Означает, что параметры отсека (координаты ограничивающего параллелепипеда, точки или коэффициенты уравнения плоскости) заданы в данной системе координат.
- **trash-->cons** – привязка поддерева конструкции к корзине.
- **trash-->object** – привязка изделия к корзине.
- **form3-->fsect** – привязка ссылочной записи отсека к заданию на расчет третьей формы.
- **sect-->fsect** – привязка ссылочной записи отсека к отсеку.

Расчеты по весовым моделям

Весовой анализ и контроль при проектировании летательных аппаратов представляет собой комплекс организационно – технических мероприятий, направленный на создание конструкции минимальной массы. Необходимым условием эффективного весового контроля является возможность получения оперативной информации о текущей массе изделия и любой его части. Весовая модель представляет всю необходимую информацию для проведения различных расчетов и представления результатов расчетов в нужной для конструктора форме.

Проведение расчетов в АСВР предусматривает получение выходных информационных форм нескольких видов. Для удобства формы различных весовых сводок занумерованы. Приведенные ниже номера форм являются исторически сложившимися в технологии проектирования (как видно ниже формы 4,5 не используются). Для удобства пользования системой в АСВР предусмотрен механизм создания заданий на расчет по разным формам с настройкой параметров расчета и состава входной/выходной информации.

Форма 1

Исходная информация – наименование конструкции корня поддерева, тип атрибута для сортировки информации из списка: классификационный признак, материал, отдел, резервный признак. Задача – выдача весовой сводки, т.е. перечень агрегатов и деталей, входящих в заданное поддерево, сгруппированный по указанному признаку и содержащий их массово-инерционные характеристики, в том числе текущие и лимитные массы.

В форму 1 выводятся дата и время расчета, наименование корня анализируемого поддерева, наименование и значение атрибута для сортировки, перечень элементов конструкции, которые выбраны по значению заданного атрибута, а также для всех элементов конструкции лимитная масса (М ЛИМ), текущая (М ТЕК), теоретическая (М ТЕОР), чертежная (М ЧЕРТ) и фактическая массы.(М ФАКТ). Кроме того, выводятся суммарные по выборке характеристики. По форме 1 можно оперативно

отслеживать превышение лимитных значений массы на всех этапах проектирования самолета.

На рис. 2 приведен пример результатов расчета по форме 1. Эта же форма позволяет контролировать отделы разработчиков и конкретных исполнителей.

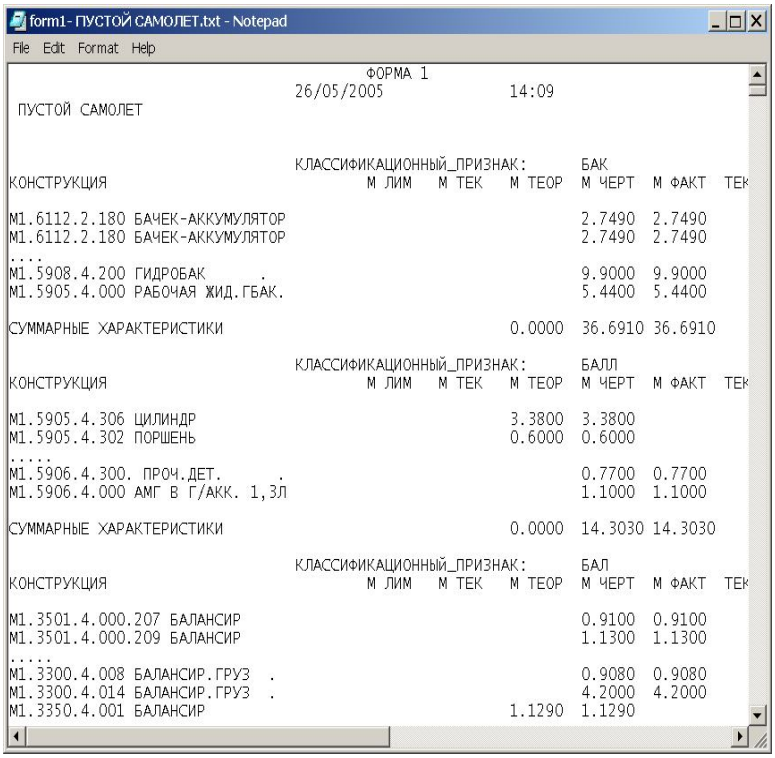


Рис. 2. Пример представления результатов расчета по форме 1

Расчет формы производится в несколько проходов по заданному поддереву конструкции. Результаты расчета формы записываются в текстовый файл на сервере. Имя файла совпадает с именем задания на расчет, расширение файла - .txt. Файл может быть передан пользователю с сервера специально предусмотренной для этого командой. Формат файла текстовый, предназначенный для импорта в MS Excell. Файл состоит из строк,

каждая строка состоит из отдельных полей, отделенных символами табуляции.

Форма 2

Форма 2 отличается от формы 1 тем, что сортировка весовой сводки осуществляется по двум указанным в задании на расчет классификационным атрибутам. В процессе просмотра весовой модели осуществляется поиск максимальных поддеревьев, в вершинах которых присутствуют непустые значения заданных атрибутов (первого и/или второго). Для каждой висячей вершины различаются четыре ситуации: 1) вершина входит в поддеревья, помеченные значениями как первого, так и второго атрибута; 2) вершина входит в поддерево, помеченное значением только первого атрибута; 3) вершина входит в поддерево, помеченное значением только второго атрибута; 4) вершина не входит ни в одно такое поддерево. Для каждого сочетания значений атрибутов (пары, по одному и ни одного) накапливается сумма масс соответствующих висячих вершин и выводится в результирующую сводку.

Форма 3

Исходная информация – наименование конструкции корня поддерева, система координат, в которой надо выдавать информацию, типа массы (текущая, теоретическая, чертежная), параметры учета подсистем (без подсистем, в составе конструкции, отдельным списком), параметры учета сосредоточенных масс (без сосредоточенных масс, в составе конструкции, отдельным списком). Задача – выдача весовой сводки, сгруппированной по отсекам, подсистемам, сосредоточенным массам.

Форма 6

Исходная информация – наименование конструкции корня поддерева, система координат, в которой надо выдавать информацию, типа массы (текущая, теоретическая, чертежная). Задача – выдача весовой сводки в древовидной форме:

Форма 7

Исходная информация – наименование конструкции корня поддерева. Задача – выдача весовой сводки в древовидной форме информацией о превышении заданных лимитных масс.

Пользовательский интерфейс

Окно работы с деревом конструкции (см рис. 2) содержит три панели: список вышестоящей иерархии конструкций с параметрами, параметры текущей просматриваемой конструкции, список непосредственных потомков текущей конструкции.

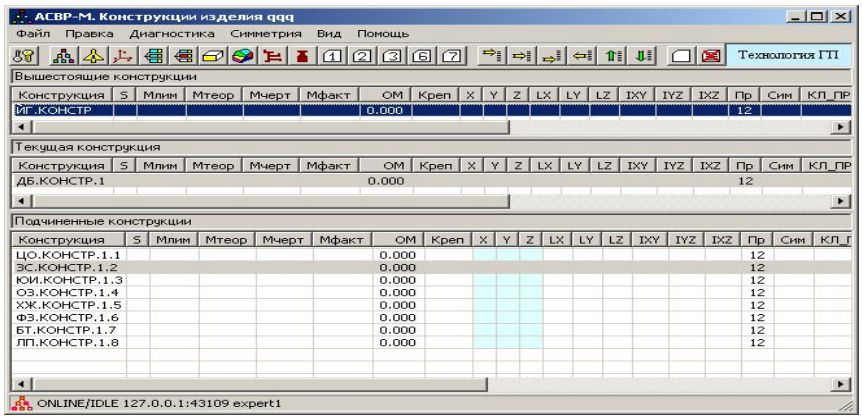


Рис. 2. Окно работы с деревом конструкции

Окно визуализации пространственных отсеков (см. рис. 3) содержит две панели: перечень отсеков и трехмерное изображение выделенных в правой панели отсеков.

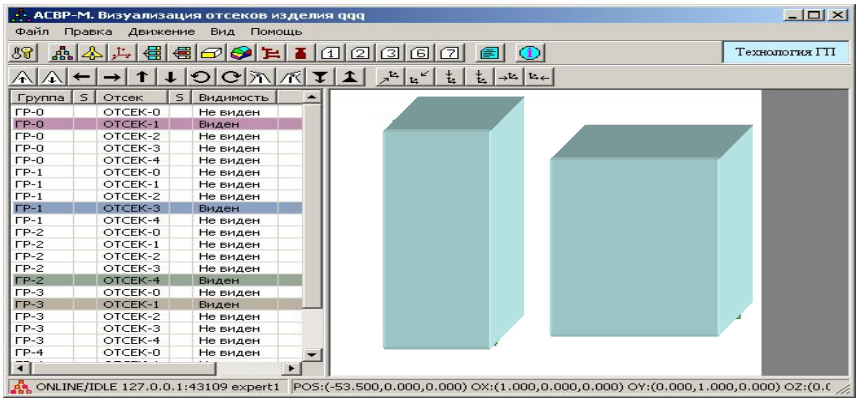


Рис. 3. Окно визуализации пространственных отсеков

Окно задания расчетов по форме 3 (рис. 4) содержит четыре панели: список заданий, список отсеков для текущего задания, список групп отсеков и список отсеков текущей группы.

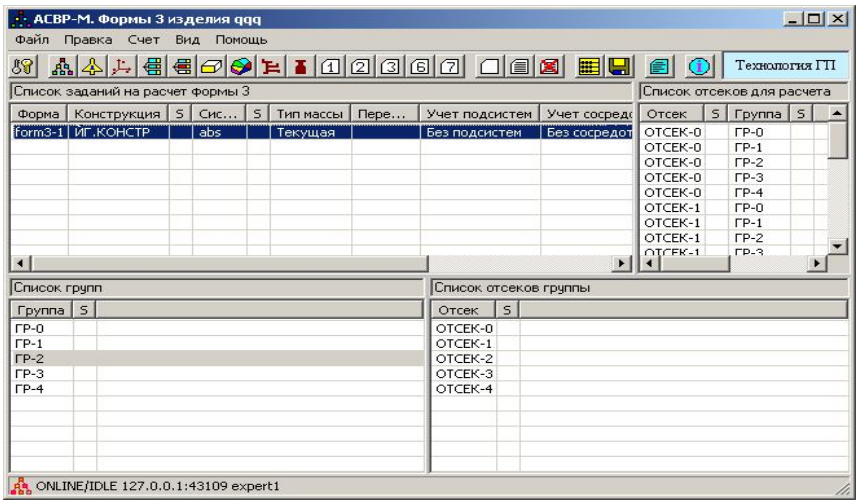


Рис. 4. Окно задания расчетов по форме 3

Заключение

Новая версия системы АСВР реализована, внедрена и активно используется. Для преемственности новой версии в системе реализован импорт файлов, выгруженных из старых баз данных VAX/VMS. Кроме того реализован импорт информации из файлов в формате DBF, а также импорт информации с характеристиками деталей и агрегатов сформированных в автоматизированной геометрической системы конструирования Unigraphics.

В заключении приведем сравнительные количественные характеристики проекта АСВР и программного комплекса на языке С автоматически сгенерированного Генератором проектов.

ПРОЕКТ (ИСХОДНОЕ ОПИСАНИЕ СИСТЕМЫ):

- Описание проекта ~ 0.6 Мб
- Количество строк в проекте ~ 21 000
- Количество процедур ~ 2000
- Количество С – функций в проекте 0

- Количество файлов проекта 105
- Количество различных окон 52
- Количество диалогов 50

СГЕНЕРИРОВАННЫЙ ТЕКСТ НА ЯЗЫКЕ С:

- Объем программного текста ~ 8.8 Мб
- Количество операторов (строк) ~ 300 000
- Количество файлов ~ 750

Из приведенных данных видно, что эффективность применения Генератора проектов в данном .проекте достаточно высока. Если ее измерять по соотношению количеству строк проекта и автоматически сгенерированного программного текста, то это мы получаем экономию более чем в пятнадцать раз, тем более что значительная часть проектных объектов не написана разработчиком, а автоматически создана генератором с использованием механизма макросов.

Литература.

1. *С.И. Скобелев, Н.И. Широков.* Весовой анализ и контроль в САПР ЛА. // Задачи и методы автоматизированного проектирования. М.: ВЦ РАН, 1991
2. *Л.Л. Вышинский, И.Л. Гринев, Ю.А. Флеров, А.Н. Широков, Н.И. Широков.* Генератор проектов – инструментальный комплекс для разработки «клиент - серверных» систем // Информационные технологии и вычислительные системы. 2003, № 1-2. С. 6-25.

АВТОМАТИЗИРОВАННАЯ СИСТЕМА БЮДЖЕТНОГО УПРАВЛЕНИЯ

Л.Л. Вышинский

В настоящей работе дается описание автоматизированной системы бюджетного управления, реализованной с помощью Генератора проектов. Бюджетное управление - это технология управления, которая позволяет руководству ставить перед своими структурными подразделениями и дочерними предприятиями конкретные производственные задачи посредством бюджетных ограничений, а также постоянного контроля над выполнением этих ограничений. Автоматизированная система бюджетного управления (АСБУ) предназначена для информационной и аналитической поддержки решения основных задач бюджетного управления. Такими задачами являются: планирование бюджетных ограничений и основных финансово–экономических показателей для дочерних предприятий, сбор информации о фактически реализованных значениях финансово–экономических показателей за отчетные периоды, анализ выполнения бюджетов и планов, составление сводных и аналитических отчетов, оценка эффективности вложения средств.

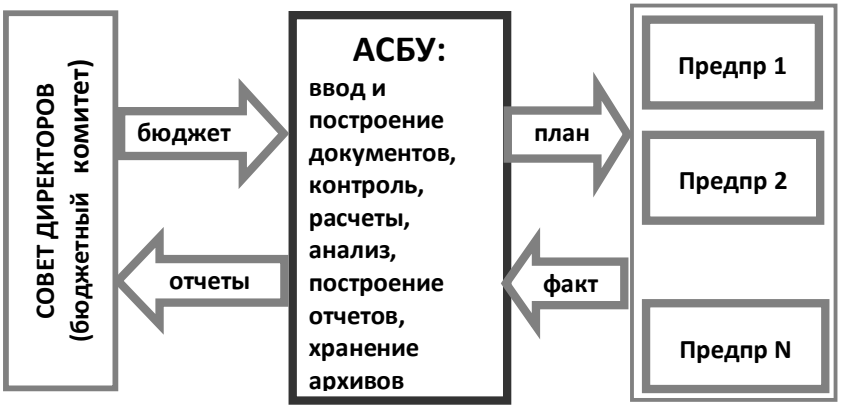


Рис. 1. Схема функционирования АСБУ в рамках корпорации

Система АСБУ реализована по технологии ГП (см. [1]), в основе которой лежит проектный подход к созданию прикладных информационных систем. Это означает, что для реализации системы создается ее проект, который представляет собой формальное описание на специальном языке и который в дальнейшем является неотъемлемой частью системы.

Центральным ядром проекта является структурно-параметрическая модель предметной области, т.е. описание основных сохраняемых информационных объектов системы. Хранение данных в АСБУ реализовано средствами реляционной СУБД. Однако структурно-параметрическая модель задает не только схему хранения в виде таблиц СУБД, но также связи между данными. На рис. 2. представлены основные объекты структурно-параметрической модели бюджетного управления.

Рис.2. Информационные объекты АСБУ

Информационная модель АСБУ, ее основные объекты и их атрибуты подробно описаны в [2].

Главной задачей АСБУ является наполнение информационной модели системы бюджетного управления актуальными данными, контроль и сохранение этих данных, решение ряда аналитических задач бюджетного управления и выпуск отчетной документации. В связи с этим, в рамках информационной модели проекта АСБУ реализовано решение следующих содержательных задач:

- Ведение журнала пользователей АСБУ.
- Ведение справочника предприятий.
- Классификация и каталогизация финансово-экономических показателей.
- Ведение каталога номенклатуры продукции выпускаемой дочерними предприятиями.
- Разработка форм внутрикорпоративных документов.
- Обеспечение формирования плановых заданий и согласования их с предприятиями.
- Сбор информации от предприятий о фактическом выполнении планов за отчетные периоды.
- Расчет сводных показателей и выпуск отчетов об исполнении планов, формирование аналитических отчетов.
- Обеспечение возможности проведения параметрических расчетов при составления планов и аналитических отчетов.
- Ведение базы данных для хранения документов и финансово-экономических показателей по всем отчетным периодам.

Первые две функции являются достаточно рутинными. Справочник предприятий имеет иерархическую структуру, а пользователи АСБУ имеют определенные права при работе в системе. Перечень возможностей пользователей (перечень привилегий) приведен в специальном разделе проекта и связан с функциями системы.

Центральным объектом бюджетного управления является каталог финансово-экономических показателей. Это кодифицированный иерархический список. Составление каталога

финансово-экономических показателей является основной задачей планово-экономических служб головного предприятия Корпорации, определяющих направления бюджетного управления.

На рис. 3. показан пример системы финансово-экономических показателей. Полная структура каталога финансово-экономических показателей может состоять из глав, разделов, параграфов. Главы, разделы, параграфы и другие структурные единицы каталогов должны объединяться по содержательному принципу. Каждый раздел (подраздел, …) должен иметь свой уникальный номер, состоящий из фиксированного набора цифр (разделы – 01, 02, …, подразделы 01, 02, …). В целях прозрачности и упорядочения структурных единиц каталога нужно правильно задать размер (длину) номера данной структуры. Главы могут именоваться с использованием буквенных символов для большей выразительности и введения мнемонических обозначений. Но, во-первых, эти мнемонические номера не должны быть слишком длинными, а во-вторых, это совсем не обязательно - иногда достаточно и цифр. Целесообразно, чтобы показатели, входящие в разделы, подразделы, параграфы наследовали номера вышестоящих структурных единиц каталога в структуре своего номера.

Каждый показатель в системе может детализироваться на произвольную глубину в соответствии с эконометрическими свойствами показателя. В АСБУ можно создавать типовые, многократно испльзуемые, структуры детализации, которые соответствуют либо общепринятым экономическим учетным категориям, либо связаны с конкретным характером деятельности предприятий корпорации. Общепринятыми структурами является классификаторы видов деятельности, стандартные перечни статей доходов и расходов, виды активов и пассивов и прочее. Специфическими структурами являются номенклатура производимой продукции, региональные структуры и реестры получателей продукции, журналы заказчиков и поставщиков. Классификаторы типовых структур детализации позволяют поддерживать определенную дисциплину нумерации финансово-экономических показателей в каталоге.

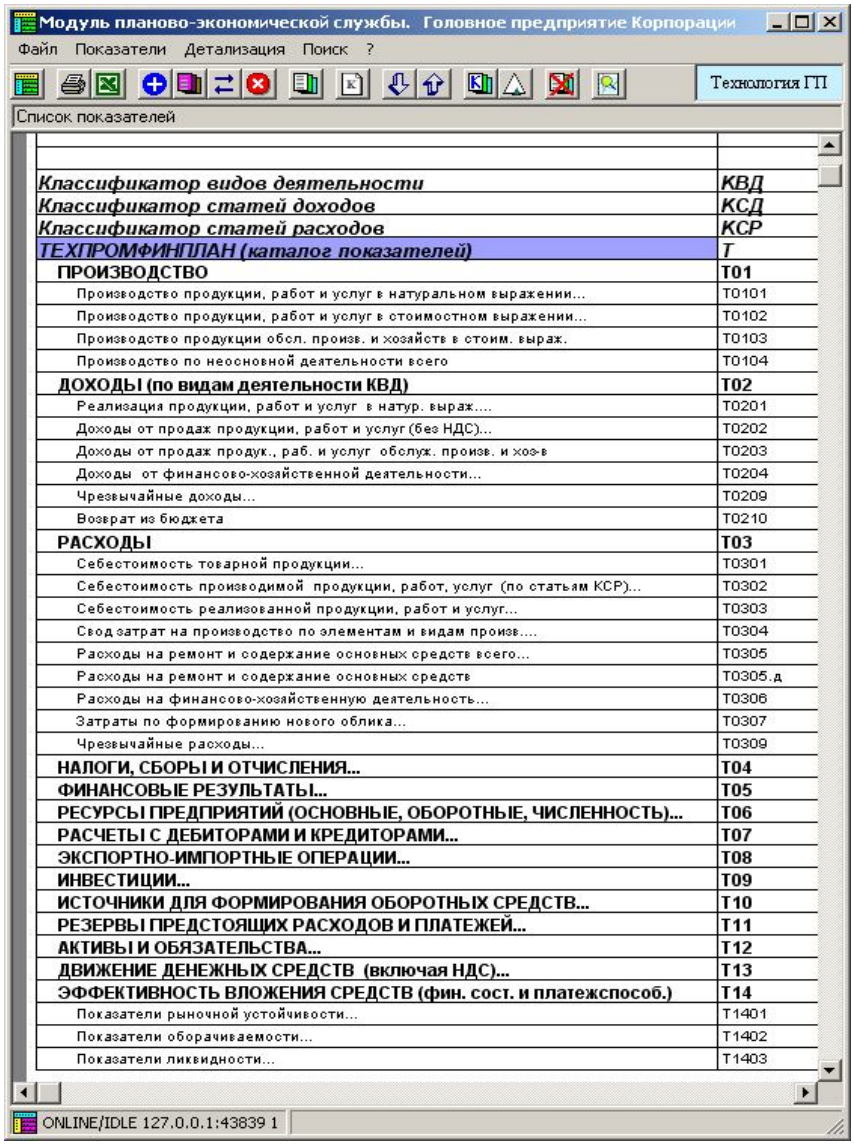


Рис. 3. Пример реализации в АСБУ каталога финансово-экономических показателей

Кроме явной детализации в структуре финансово-экономических показателей может использоваться неявная

(«виртуальная») детализация показателей. "Виртуальная" детализация задается с помощью ссылок на другие структуры каталога - или типовые структуры расшифровки показателей. Например, на типовую детализацию номенклатуры производимой продукции, на типовую структуру статей расходов и доходов и прочее. На рис.4 дана иллюстрация принципа «виртуальной» детализации.

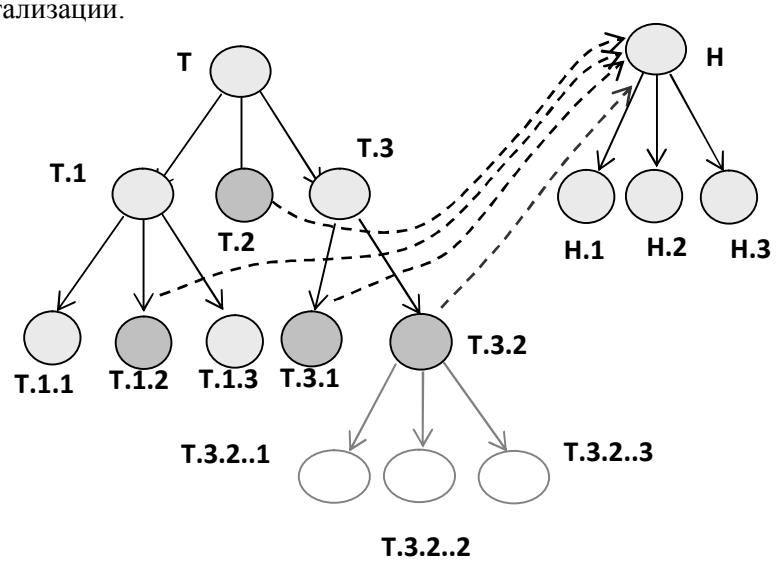


Рис. 4. «Виртуальная» детализация

Здесь схематически пунктирными стрелками показаны ссылки из структуры каталога **Т** на типовую структуру **Н**, которая показывает одинаковую схему расшифровки для ряда показателей. При «виртуальной» детализации в каталоге явно подчиненные структуры не создаются, но в дальнейшем при решении различных задач могут использоваться показатели с номерами, динамически образуемыми по указанным ссылкам. Такие динамически образованные показатели могут храниться в базе данных, фигурировать в плановых, отчетных и аналитических формах. «Виртуальная» детализация избавляет от необходимости описывать всевозможные комбинации детализаций, а задет механизм динамического расширения совокупности показателей.

Совокупность явных и виртуальных финансовых показателей образуют эконометрическое пространство данных, в рамках которого в АСБУ осуществляется управление финансовой и экономической деятельностью корпорации. Финансово-экономические показатели не являются независимыми и связаны множеством экономических соотношений и ограничений. Эти соотношения так же, как и само множество показателей, могут изменяться в течение времени и в связи с возникновением новых управленческих задач. Поэтому в АСБУ реализованы механизмы задания пользователями широкого класса аналитических соотношений. Эти соотношения задаются непосредственно при построении каталога или в шаблонах плановых, отчетных и аналитических форм. Но о формах будет сказано ниже. Что же касается процедуры создания новых показателей, то она реализована с использованием специального диалога, в котором задаются необходимые реквизиты (см. рис. 5).

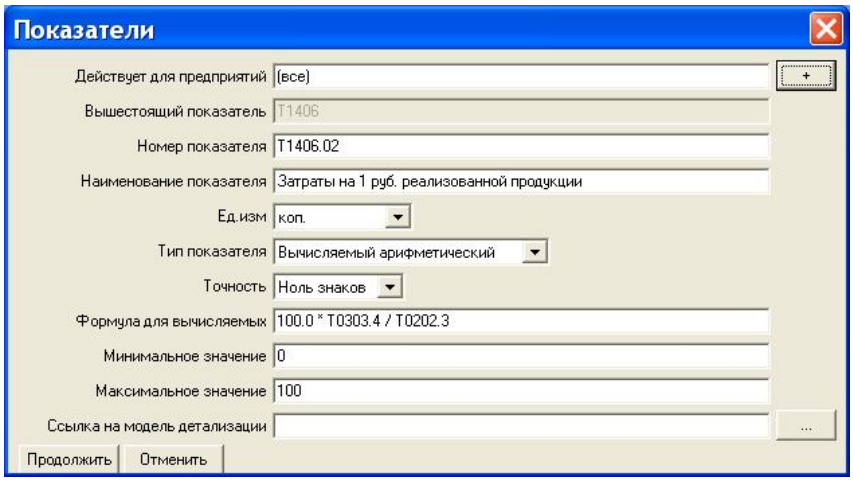


Рис. 5. Ввод реквизитов показателей

Показатели могут иметь тип, как на рис. 5, вычисляемого по формуле, могут быть просто вводимыми числовыми или текстовыми значениями, а могут быть синтетическими, т.е. вычисляемыми как сумма подчиненных показателей.

Ввод первичных данных, т.е. значений показателей, производится на основании директивных, либо бухгалтерских, производственных и других документов, содержащих нужную информацию. На рис. 6. показан пример документооборота, который связан с бюджетным управлением корпорацией.

Рис. 6. Документооборот в бюджетном управлении

Совокупность плановых, отчетных и аналитических документов в АСБУ организована в альбом шаблонов форм, который состоит из ряда вложенных групп шаблонов (см. Рис. 7). Группы форм ориентированы на решение разных задач бюджетного управления – планирование, контроль, прогнозирование, анализ, моделирование и прочее. В свою очередь формы, входящие в группу, отражают ту или иную частную сторону производственной или экономической деятельности предприятия, которые подвергаются планированию, контролю, анализу.

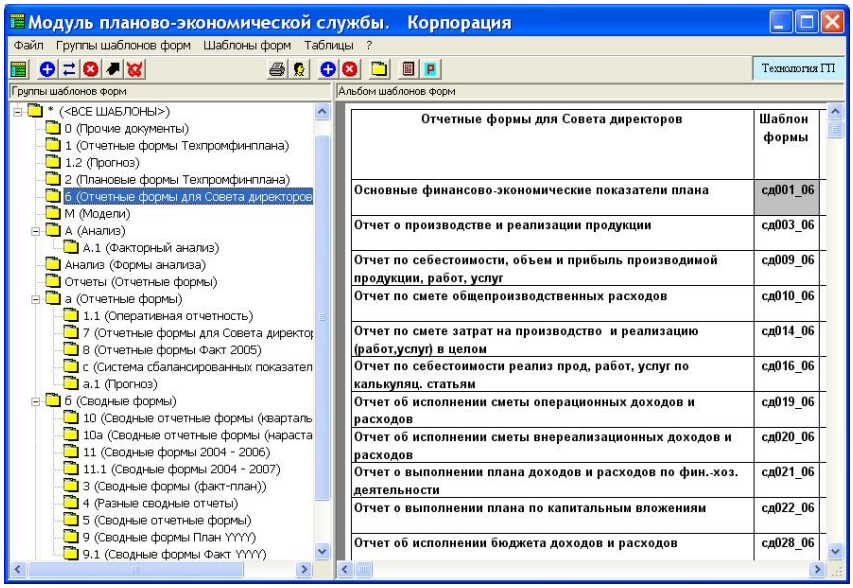
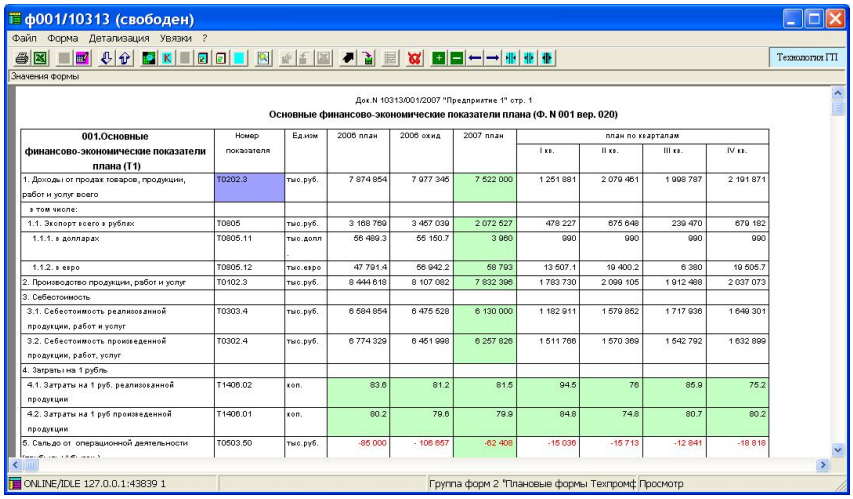


Рис. 7. Альбом шаблонов форм

Формы, таблицы, строки, столбцы и поля обязательно привязаны к определенному предприятию, определенному плановому или отчетному периоду и статусу показателя (плановый или фактически реализованный). Поля форм (т.е. показатели) могут быть увязаны между собой целым рядом аналитических соотношений. В АСБУ реализован «мастер» форм, который позволяет строить весьма сложные конструкции, включающие не только описание содержательных задач, решаемых с помощью данных форм, но и наглядное их представление, стили, цвет шрифтов и заливок полей, выравнивание и прочее. Среди основных реквизитов средствами «мастера» форм вводятся соотношения, которые увязывают данные как одной формы, так и нескольких форм. Таким образом, формы – это не только средство ввода данных, но и способ вычисления вторичных (вычисляемых) показателей. Вместе с формулами, заданными в каталоге, соотношения заданные в шаблонах форм описывают множество допустимых значений финансово-экономических показателей.

Суть бюджетного управления в данном контексте состоит в задании и вычислении плановых показателей из множества допустимых значений, ввод и вычисление фактически реализованных показателей и вычисление сводных показателей. Все это осуществляется посредством создания и редактирования документов, формы которых описаны в альбоме шаблонов. Множества показателей, представленных в разных формах, могут пересекаться между собой, поэтому при построении документов могут возникнуть противоречия и конфликты. Для этого в АСБУ существуют специальные механизмы, показывающие непосредственно в формах имеющиеся конфликты и позволяющие устранять их. После устранения выявленных противоречий документ может быть принят, а его показатели занесены в реестр.

На рис. 8 приведен пример заполненной формы.



001.Основные финансово-экономические показатели плана (Т1)	Номер показателя	Ед.изм	2005 план	2006 ожид	2007 план	план по кварталам			
						I кв.	II кв.	III кв.	IV кв.
1. Доходы от продаж товаров, продукции, работ и услуг всего	T0202.3	тыс.руб.	7 874 854	7 977 345	7 522 000	1 251 881	2 079 461	1 998 787	2 191 871
в том числе:									
1.1. Экспорт всего в рублях	T0805	тыс.руб.	3 168 769	3 457 039	2 072 527	478 227	675 648	239 470	679 182
1.1.1. в долларах	T0805.11	тыс.долл	56 489.3	55 150.7	3 980	990	990	990	990
1.1.2. в евро	T0805.12	тыс.евро	47 791.4	56 942.2	58 793	13 507.1	19 400.2	6 380	19 505.7
2. Производство продукции, работ и услуг	T0102.3	тыс.руб.	8 444 618	8 107 082	7 832 396	1 783 730	2 099 105	1 912 488	2 037 073
3. Себестоимость									
3.1. Себестоимость реализованной продукции, работ и услуг	T0303.4	тыс.руб.	6 584 854	6 475 528	6 130 000	1 182 911	1 579 852	1 717 936	1 649 301
3.2. Себестоимость произведенной продукции, работ, услуг	T0302.4	тыс.руб.	6 774 329	6 451 998	6 257 826	1 511 766	1 570 369	1 542 792	1 632 899
4. Затраты на 1 рубль									
4.1. Затраты на 1 руб. реализованной продукции	T1406.02	коп.	83.6	81.2	81.5	94.5	76	85.9	75.2
4.2. Затраты на 1 руб произведенной продукции	T1406.01	коп.	80.2	79.6	79.9	84.8	74.8	80.7	80.2
5. Сальдо от операционной деятельности	T0503.50	тыс.руб.	-85 000	- 106 657	-62 408	-15 036	-15 713	-12 841	-18 818

Рис. 8. Плановая форма основных финансово-экономических показателей

Выделенные цветом поля являются вычисляемыми, остальные вводятся вручную или загружаются автоматически из реестра и из других документов.

Основным хранилищем финансово-экономической информации является реестр показателей. В реестр данные попадают из документов. Хранятся плановые и фактические данные для каждого показателя в разрезе предприятий и периодов. На основании данных реестра строятся сводные отчеты по корпорации, а также решаются различные аналитические задачи, строятся прогнозы, ищутся оптимальные варианты будущих плановых заданий и бюджетных ограничений. Для целей анализа в АСБУ реализован механизм описания и проведения параметрических и оптимизационных расчетов. На рис. 9 приведен пример описания модели параметрических расчетов уровня затрат на рубль дохода в зависимости от себестоимости выпускаемой продукции и оценка возможного минимального значения этого важного экономического показателя производства и реализации продукции.

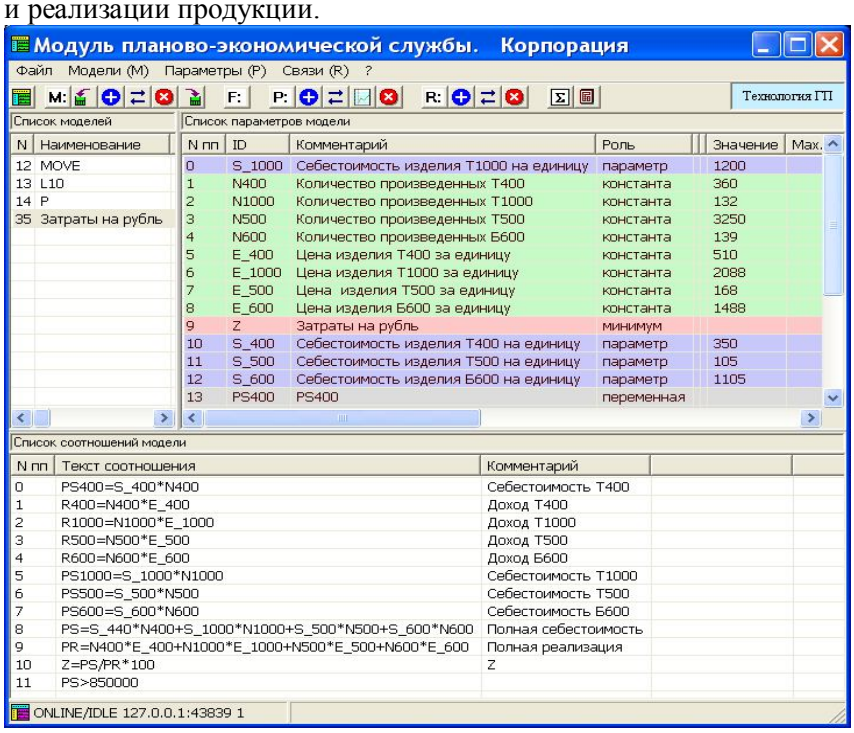


Рис. 9. Пример модели параметрических расчетов

Программный комплекс АСБУ, созданный с помощью Генератора проектов, состоит из описания проекта на языке ЭЗОП, текста на языке С – результата работы ГП и исполняемых программ бизнес-сервера, модуля планово-экономической службы, модуля системного администратора и ряда служебных библиотек. Ниже приведены количественные характеристики объема программного комплекса:

ПРОЕКТ (ИСХОДНОЕ ОПИСАНИЕ СИСТЕМЫ):

- Описание проекта ~ 2 Мб
- Количество строк в проекте ~ 70 000
- Количество функций ~ 1500
- Количество С – функций в проекте ~ 300
- Количество файлов проекта ~ 200

СГЕНЕРИРОВАННЫЙ ТЕКСТ НА ЯЗЫКЕ С:

- Объем программного текста ~ 16.5 Мб
- Количество операторов (строк) ~ 500 000
- Количество С - функций ~ 20 000
- Количество файлов ~ 1000

Система АСБУ была разработана по заказу топливной компании ОАО «ТВЭЛ» и функционирует на всех производящих и инфраструктурных предприятиях компании с 2005 г.

Литература

1. *Л.Л. Вышинский, И.Л. Гринев, Ю.А. Флеров, А.Н. Широков, Н.И. Широков.* Генератор проектов – инструментальный комплекс для разработки «клиент - серверных» систем // Информационные технологии и вычислительные системы. 2003, № 1-2. С. 6-25.
2. *Л.Л. Вышинскийю* Информационная модель и архитектура автоматизированной системы бюджетного управления предприятием. М.: ВЦ РАН, 2007. 39 с.

ИСПОЛЬЗОВАНИЕ ТЕХНОЛОГИИ ГЕНЕРАТОР ПРОЕКТОВ ДЛЯ СОЗДАНИЯ ЛАБОРАТОРНЫХ СЕТЕВЫХ АУКЦИОНОВ

С.А. Скиндерев

Введение

Для проведения сетевых аукционов используются различные механизмы взаимодействия его участников [1]. Первый механизм – это непрерывный двойной аукцион. Основные требования этого механизма – способность работы системы в реальном времени и наличие удобного пользовательского интерфейса. Работа в реальном времени подразумевает возможность без задержек обрабатывать входные сигналы и отображать состояние торговой сессии, а удобный пользовательский интерфейс необходим для минимизации невынужденных ошибок участников аукциона.

Второй – аукцион с диспетчером. Основной особенностью сетевого аукциона с диспетчером является использование механизма централизованного сбора закрытых заявок программой-диспетчером, которая решает задачу максимизации некоторого заданного функционала выигрыша.

Третий – аукцион с наведенными заявками [2, 3]. Сетевой аукционный механизм с наведенными заявками по отношению к предыдущим можно назвать компромиссным вариантом, сочетающим в себе прозрачность ценообразования и организованность взаимодействия участников аукциона. Основная идея аукциона с наведенными заявками состоит в том, что для каждого агента совокупность заявок всех остальных агентов порождает для него встречную заявку. В основе работы данного торгового механизма лежит вычисление для каждого агента наведенных заявок на основании его собственной заявки и заявок остальных участников аукциона. Прозрачность ценообразования сохраняется, поскольку всегда можно либо согласиться с наведенной заявкой, либо изменить свою, продолжая торг с виртуальным партнером. Роль диспетчера в данном механизме сводится к корректному распространению наведенных заявок.

Аукцион с наведенными заявками, сочетая в себе плюсы остальных механизмов, предъявляет более жесткие требования к программной реализации. С одной стороны, он требует в режиме реального времени обрабатывать сетевые запросы и производить сложные рыночные расчеты, а с другой - обеспечивать удобный пользовательский интерфейс.

Программный комплекс для проведения сетевых аукционов обычно состоит из прикладного сервера (диспетчера) и множества клиентских модулей участников аукциона. Также в некоторых случаях может быть некоторый управляющий модуль (который может быть совмещен с управляющим сервером). Такая архитектура укладывается в базовую модель Генератора проектов (ГП, см. [4]), причем сильными сторонами ГП являются обеспечение быстрого и надежного сетевого взаимодействия, а также достаточно низкоуровневая работа с ресурсами ЭВМ, позволяющая производить сложные расчеты в режиме реального времени.

Технически работа диспетчера заключатся в принятии потока заявок от участников аукциона, перерасчета состояния рынка и выдачи ответа всем участникам. Такая модель аукциона легко реализуется с помощью штатных средств ГП – запросов и оповещений. Еще одна особенность механизма с наведенными заявками состоит в том, что все заявки участников должны поступать на обработку строго последовательно. Технология ГП позволяет удовлетворить этому требованию. Сервер устроен таким образом, что вся прикладная логика его работы реализуется в одном потоке.

Единственным слабым местом ГП в реализации сетевого аукциона являются ограниченные возможности по созданию встроенного пользовательского интерфейса. Но технология ГП предоставляет программный интерфейс для разработки внешних модулей, интегрируемых в систему. Для лабораторных аукционов удобный и привычный пользовательский интерфейс был реализован силами студентов (участниками экспериментов) и встроен в систему средствами ГП.

Наличие встроенного в ГП лексического анализатора позволило создать язык описания сетевых аукционов, тем самым позволяя с помощью разработанной системы проводить широкий

спектр экспериментов, модель которых укладывается в аукцион с наведенными заявками.

Общая модель программного комплекса

Основными участниками аукциона являются диспетчер и игроки. Основная функция диспетчера – обеспечение взаимодействия игроков между собой при соблюдении бизнес - логики в соответствии с теоретико-игровой моделью.

В соответствии с принятой в ГП архитектурой (клиент-сервер) функции диспетчера выполняет прикладной бизнес-сервер, а клиентские модули раздаются игрокам. Также выделяется еще один клиентский модуль для управления и мониторинга за ходом торгов.

Сетевое взаимодействие между модулями в ГП реализовано таким образом, что прикладная логика работы бизнес-сервера осуществляется в одном потоке, что позволяет обрабатывать запросы строго последовательно. Это согласуется с одним из главных постулатов механизма: все заявки участников должны поступать на обработку строго последовательно.

Подсистема информационной безопасности позволяет без искажений передавать данные в сети, в том числе и глобальной. Наличие системы аутентификации позволяет привязывать каждого игрока к его логину и не терять контекст аукциона даже при возможных разрывах связи, а также оперативно передавать их игроку после успешной повторной аутентификации.

Описание аукциона с наведенными заявками

Основными компонентами аукциона с наведенными заявками являются:

- игроки;
- роли игроков;
- позиции игроков на рынке (агенты);
- проекты;
- рынки (узлы проектов);
- множество заявок, поданных игроками (агентами) и диспетчером.

Работа аукциона состоит из нескольких торговых сессий (попыток), каждая из которых является независимым экономическим экспериментом. Постоянными остаются множество игроков и экономическая структура аукциона.

Основные действующие лица в работе аукциона – это экономические агенты (множество $A = \{1, \text{K}, N\}$), которых можно разделить на две группы: покупатели товаров или услуг и продавцов. Продавцы характеризуются своими затратами на производство одной единицы товара или услуги $c_i (i \in A)$, а покупатели – выкупной стоимостью $v_i (i \in A)$ – ценой, по которой они поставляют товар конечным потребителям. Продавцы, реализуя единицу товара по цене p_i^s, получают выигрыш $u_i = p_i^s - c_i$. Покупатели получают $u_i = v_i - p_i^B$.

Множество агентов разбивается на подмножества ролей игроков. Таким образом, каждый игрок действует от лица своих агентов, максимизируя свой суммарный выигрыш. В каждой торговой сессии множество ролей совпадает с множеством игроков. При смене торговой сессии игроки могут меняться ролями по некоторым заданным правилам.

Основные структурные объекты аукциона – это узлы. В каждом узле находится один или несколько агентов одного типа. Узел представляет собой рынок с непрерывным двойным аукционом. С одной стороны, играет один или несколько агентов, подавая простые заявки на покупку (bid) или продажу (ask), а с другой – диспетчер, который из множества имеющихся простых заявок формирует встречные наведенные.

Проекты – это множество подмножеств множества узлов. Проекты являются основой для совершения сделки. Для того чтобы произошла сделка на проекте, необходимо и достаточно, чтобы сумма заявок агентов-покупателей была не меньше суммы заявок агентов-продавцов.

Позиции игроков на рынке определяются ролями в начале торговой сессии, в течение которой позиция игрока совпадает с состоянием соответствующего агента.

В ходе торгов игроки формируют поток заявок диспетчеру, который каждую такую заявку должен обработать, вычислив новое состояние аукциона, и разослать его всем игрокам. Для подачи заявок в технологии ГП используются запросы (request), а рассылка актуального состояния аукциона осуществляется с помощью оповещений (notification).

Всю структура аукциона можно представить в виде нескольких многосвязных списков, для чего в ГП используются базовые структуры данных – документы (document).

Подробное описание сущностей

Информация об игроках делится на две части. Первая отражает постоянную информацию об игроке: ФИО, торговый номер, привязка к логину для связи с аутентификационными параметрами рабочей сессии клиента. Вторая содержит текущие параметры игрока в рамках торговой сессии: привязка к роли в данной попытке, его выигрыши (текущий и суммарный) и при необходимости дополнительные параметры сессии (например, приватные параметра игрока в соответствии с ролью).

Роль игрока не содержит никакой дополнительной информации, а служит только для корректного распределения позиций в начале торговой сессии.

Узел содержит информацию о типе рынка (продавцы или покупатели). Из узлов составляются проекты. Между этими сущностями связь «многие ко многим».

Позиции игроков связывают текущие роли игроков с узлами (многие ко многим). Содержат информацию о начальных запасах (или лимите потребления) продукта, о затратах (или выкупной стоимости) и о выигрыше агента в текущей торговой сессии.

Заявки привязываются к рынку (узлу) и к игроку. Содержат информацию о номинале заявки и своей кратности (заявка может быть подана агентом на реализацию не одной единицы, а сразу на несколько по единой цене). Для наведенных заявок, сформированных диспетчером, дополнительно хранится ссылка на проект, так как один узел может входить в состав нескольких проектов.

Язык описания проектных игр

Для инициализации аукциона необходимо описать все параметры аукциона в рамках теоретико-игровой модели проектной игры. Для этого был разработан язык описания проектных игр. Для разбора описания проекта используется встроенный в ГП лексический анализатор.

Для описания проектной игры необходимо описать все необходимые сущности: узлы, проекты, связи между узлами и проектами, роли игроков и позиции игроков на узлах. Для задания узлов необходимо указать их идентификаторы и типы. Для определения проектов достаточно указать только идентификаторы. Связи между узлами и проектами задаются парами идентификаторов. При описании ролей определяются их идентификаторы и кратности (для создания нескольких одинаковых ролей достаточно описать ее один раз и указать количество). Для задания позиций необходимо указать идентификаторы узлов и ролей, а также начальный массив затрат или выкупных стоимостей. Для создания аукциона, который состоит из нескольких одинаковый независимых аукционов, можно задать его кратность и не описывать его несколько раз в файле.

Вообще говоря, затраты (выкупные стоимости) агентов не являются постоянными величинами и могут меняться в разных торговых сессиях. Для этого предусмотрено описание переменных, поэтому наборы затрат (выкупных стоимостей) задаются не числовыми параметрами, а переменными. Переменные в системе трех типов: временные ряды, случайные величины и вычисляемые. Временные ряды задаются перечислением числовых значений и меняются циклически при переходе от попытки к попытке. Случайные переменные определяются диапазоном и в начале каждой попытки принимают произвольные значения из указанного диапазона. Вычисляемые переменные задаются арифметическими выражениями, содержащими числовые значения и описанные ранее переменные.

Аукцион с наведенными заявками подразумевает, что количество игроков (ролей) строго фиксировано. Например, есть игра четырех лиц, и она полностью описана в файле, а

планируется провести эксперимент с двенадцатью участниками. Тогда необходимо поставить кратность аукциона три, и в начале каждой попытки система будет разбивать участников на независимые четверки по заданному правилу. Также, поставив кратности ролей, можно сделать, чтобы все двенадцать человек играли на едином аукционе, причем игроки с одинаковыми ролями будут конкурировать между собой на соответствующих рынках. Чтобы подкорректировать числовые параметры, также нет необходимости редактировать весь файл. Достаточно просто изменить значения соответствующих переменных.

Результаты торгов

Итогом торгов является набор колоночных файлов с результатами торговых сессий. Встроенная в ГП система протоколирования и формирования журналов позволяет формировать необходимые файлы заданного формата.

Итоговый протокол состоит из трех файлов. Первый файл содержит информацию о ходе торгов. В каждой строке пишется номер попытки, время события, тип события (подача заявки или совершение сделки), идентификатор игрока, идентификатор рынка, номинал заявки или цена сделки, кратность заявки или сделки, выигрыш агента (в случае совершения сделки), а также текущий спрос (bid) и предложение (ask) на данном рынке.

Второй файл содержит информацию в разрезе агентов: номер попытки, идентификатор игрока, идентификатор рынка, частные параметры (затраты или выкупные стоимости), выигрыш агента.

В третий файл записывается информация в разрезе игроков: номер попытки, идентификатор игрока, ФИО, торговый номер, идентификатор роли, выигрыш в игрока в текущей попытке и суммарный выигрыш.

Первый файл пишется в течение торговой сессии по фактам возникновения событий, второй – при открытии и закрытии торговой сессии, а третий – только при закрытии.

Пользовательский интерфейс

Торги на сетевом аукционе проходят в режиме реального времени, поэтому система должна предоставлять игрокам

возможность быстрого реагирования на изменения состояния рынков. Следовательно, пользовательский интерфейс должен быть максимально привычен и удобен игрокам, чтобы минимизировать количество невынужденных ошибок, связанных с неоптимальным расположением элементов управления в клиентском модуле.

Технология ГП ориентирована на работу с базами данных и не предоставляет гибкого интерфейса для взаимодействия с пользователем (игроком). Однако она позволяет создавать программный интерфейс для стороннего разработчика посредством внешнего СИ-проекта. При этом формируется динамически загружаемая библиотека и соответствующий заголовочный файл (для языков C, C++ или C#).

Встроенная в ГП система поддержки версий позволяет контролировать целостность всех компонент системы и упрощает работу со сторонними разработчиками, позволяя быстро скомпилировать очередную версию программного комплекса и собрать инсталляционный пакет.

Универсальность интерфейса

Работа клиентского модуля в конечном итоге сводится к отправке заявок и получению состояния рынка. Библиотеку интерфейса можно использовать не только для интеграции с внешними клиентскими оболочками, но и для написания автономных программных клиентов. Таким образом, можно сформировать некоторый алгоритм поведения робота на аукционе. Таких роботов можно использовать для расширения класса проводимых экспериментов. Также с их помощью можно запускать эксперименты без привлечения реальных игроков, получать предварительные результаты и при необходимости корректировать начальные данные аукциона.

Заключение

Данный программный комплекс используется для проведения экспериментов в лаборатории экспериментальной экономики МФТИ (ГУ). Использование технологии Генератор проектов позволило в короткие сроки создать цельную систему силами одного разработчика. Также технология позволяет в

течение длительного времени поддерживать и развивать систему, тем самым расширяя класс проводимых экспериментов.

Как показала практика, комплекс позволяет проводить эксперименты, в которых количество участников достигает двадцати, причем скорость подачи заявок местами может доходить до нескольких в секунду от одного игрока. При этом система успевает обрабатывать эти заявки, перерассчитывать состояние рынка и рассылать ответы за время, позволяющее игрокам без видимых задержек реагировать на изменения состояния аукциона.

Литература

1. Меньшиков И.С., Платонов В.В., Скиндерев С.А., Чабан А.Н. Сравнительный анализ эффективности лабораторных сетевых аукционов. М.: ВЦ РАН, 2007.

2. Журавель Ю.Ю., Меньшиков И.С. Двойной аукцион для сетевых рынков. М.: ВЦ РАН, 2003.

3. Журавель Ю.Ю., Меньшиков И.С. Поведение сетевого двойного аукциона с учетом потерь при транспортировке. Сообщения по прикладной математике. М.: ВЦ РАН, 2004.

4. Вышинский Л.Л., Гринев И.Л., Флеров Ю.А., Широков А.Н., Широков Н.И. Генератор проектов – инструментальный комплекс для разработки “клиент-серверных” систем. // Информационные технологии и вычислительные системы. 2003, N 1-2. М. С. 6-24.

СОДЕРЖАНИЕ