УДК 681.3

АРХИТЕКТУРА КЛИЕНТСКОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ WEB-ПРИЛОЖЕНИЙ, ОРИЕНТИРОВАННЫХ НА ПРЕДСТАВЛЕНИЕ ДАННЫХ

О. Б. Арушанян[1], Н. А. Богомолов[1], А. Д. Ковалев[1], М. Н. Синицын[1]

Обсуждается подход к созданию WEB-приложений, ориентированных на представление данных, который основан на существенном использовании клиентского программного обеспечения (ПО), работающего в среде стандартного Интернет-браузера. Рассматриваются преимущества и недостатки данного подхода. Приведена архитектура клиентского ПО, его программная объектная модель, иерархия классов, модель прикладных программных событий. Описана организация динамической загрузки с сервера программ и данных. Работа выполнена при поддержке РФФИ (гранты № 02–07–90236 и № 04–07–90288).

Ключевые слова: WEB-приложения, клиентское программного обеспечения, Интернет-браузеры, объектная модель, динамическая загрузка.

1. Введение. Как правило, программное обеспечение WEB-приложений, ориентированных на представление данных, имеет двухуровневую клиент-серверную организацию. В качестве универсальной клиентской части приложения выступает стандартный Интернет-браузер. Серверные компоненты ПО выполняют основную бизнес-логику приложения и оформляют результаты своей работы в стандартных Интернет-форматах. HTML-файлы используются для текстового представления данных и для оформления элементов пользовательского интерфейса. Для графического представления данных используются растровые изображения в форматах JPEG и GIF.

По мере повышения производительности рабочих станций получает распространение подход к организации WEB-приложений, в котором стандартный Интернет-браузер дополняется специальными программными компонентами, реализованными в технологиях Java, ActiveX, JavaScript, VBScript. Это дополнительное клиентское ПО может выполнять существенную часть бизнес-логики приложения, связанной с организацией пользовательского интерфейса и отображением данных. В этом случае серверные компоненты ПО, в основном, осуществляют доступ к информации, хранящейся в серверных базах данных.

Главными задачами дополнительного клиентского ПО для WEB-приложений, ориентированных на представление данных, являются:

— обеспечение интерфейса пользователя (выполнение сценариев, активизируемых пользователем с помощью элементов управления);

— накопление данных, передаваемых на клиентский компьютер в процессе работы приложения;

— динамическая генерация визуальных представлений данных.

Наличие дополнительного клиентского ПО позволяет WEB-приложению улучшить ряд эксплуатационных характеристик:

— уменьшить нагрузку на сервер,

— уменьшить объем передаваемых по сети данных,

— уменьшить время реакции на команды пользователя.

Вычислительная нагрузка на сервер снижается за счет переноса на компьютер клиента работы по реализации части бизнес-логики приложения, связанной с организацией пользовательского интерфейса и визуализацией данных.

Поскольку часть воздействий пользователя отрабатывается непосредственно на компьютере клиента, то уменьшается количество обращений к серверу, снижается объем передаваемых по сети данных.

Визуализация на стороне клиента также снижает объем передаваемой по сети информации. Это происходит за счет того, что вместо пересылки по сети готовых визуальных представлений можно ограничиться передачей исходных данных, которые, как правило, имеют существенно меньший объем. Дополнительного уменьшения объема передаваемой по сети информации можно добиться за счет накопления на стороне клиента полученных с сервера исходных данных, что позволяет избежать повторной пересылки ранее переданной информации.

[1] Научно-исследовательский вычислительный центр, Московский государственный университет им. М. В. Ломоносова, 119992, Москва; e-mail: arush@srcc.msu.su, kovalev@srcc.msu.su

Обращения к серверу и ожидание очередных порций информации вносят существенные задержки в работу WEB-приложения. Благодаря уменьшению количества обращений к серверу и снижению объема передаваемых по сети данных уменьшается и время реакции WEB-приложения на команды пользователя, что способствует повышению комфортности его работы.

Однако подход к созданию WEB-приложений, предусматривающий наличие развитого дополнительного клиентского ПО, имеет и ряд недостатков. Во-первых, увеличивается время "раскрутки" WEB-приложения, так как модули дополнительного клиентского ПО должны передаваться на компьютер клиента в начале каждого сеанса работы. Это является главным недостатком данного подхода. Во-вторых, реализация развитого дополнительного клиентского ПО является достаточно трудоемкой задачей в силу того, что при его создании необходимо учитывать особенности функционирования ПО на различных программно-аппаратных платформах компьютера клиента. Как правило, WEB-приложения, содержащие развитые компоненты дополнительного клиентского ПО, поддерживают ограниченное число программно-аппаратных платформ.

Несмотря на приведенные недостатки, преимущества использования дополнительного клиентского ПО способствуют все более широкому его применению, продолжают активно развиваться технологии создания дополнительного клиентского ПО: Java, JavaScript, VBScript.

Целесообразно выделить класс WEB-приложений, ориентированных на представление данных, бизнес-логика которых позволяет вообще отказаться от использования серверных промышленных баз данных для доступа к публикуемой информации. В этом случае дополнительное клиентское ПО полностью обеспечивает функцию доступа к данным за счет организации базы данных на стороне клиента. При этом сами публикуемые данные размещаются на сервере в специальным образом организованных файлах, играющих роль хранилища данных. В начале работы приложения на клиентский компьютер передается информация о размещении фрагментов данных в хранилище. Файлы из хранилища, содержащие требуемые данные, передаются на клиентский компьютер в процессе работы приложения по запросам пользователей. Организованные подобным образом WEB-приложения могут работать не только в Интернет, но и на автономных, не подключенных в сеть компьютерах. Отсутствие серверных компонент позволяет осуществлять распространение подобных WEB-приложений без каких-либо изменений на внешних носителях информации (дискеты, CD и др.).

2. Архитектура клиентского ПО

2.1. Общие замечания. При создании клиентского ПО были поставлены следующие задачи:

— обеспечение надежной работы ПО в среде наиболее распространенных современных Интернет-браузеров;

— минимизация начального объема информации, передаваемой на компьютер клиента в начале сеанса работы WEB-приложения;

— разработка открытой модульной архитектуры клиентского ПО, допускающей развитие имеющихся и добавление новых средств визуализации данных;

— формализация системы управления интерфейсом пользователя WEB-приложения с целью обеспечения его параметризации в конструкторах электронных публикаций данных.

Для обеспечения надежной работы в большинстве современных Интернет-браузерах при создании компонент дополнительного клиентского ПО был использован язык JavaScript. Опыт создания компонент клиентского ПО на языке Java [1, 2] показал наличие ряда проблем, возникающих при их использовании: отличие версий Java-машин различных производителей, проблемы их совместимости с разными программно-аппаратными платформами клиентских компьютеров, отсутствие средств поддержки Java в стандартной поставке некоторых браузеров. Использование языка JavaScript [6] было обусловлено еще и тем, что он поддерживается практически всеми современными Интернет-браузерами, а формальная спецификация языка стандартизирована Европейской ассоциацией производителей компьютеров ECMA [5]. Кроме того, язык JavaScript тесно интегрирован с активно развиваемой и стандартизированной консорциумом W3C объектной моделью документа DOM. Благодаря этому язык JavaScript хорошо подходит для создания многооконных WEB-приложений, в качестве универсальной клиентской части которых выступает стандартный Интернет-браузер.

Для размещения на страницах HTML-документов динамически изменяемых визуальных графических представлений данных, как правило, используются программные надстройки (Java, Flash, ActiveX и т.д.). В [4] рассмотрена независимая от программно-аппаратной платформы компьютера клиента техника динамической генерации в среде стандартного Интернет-браузера параметризованных содержательными данными изображений на основе растровых шаблонов. Эта техника позволяет создавать динамически изменяемые изображения только средствами стандартного HTML и JavaScript, без привлечения упомянутых

выше специализированных программных надстроек.

Необходимость разработки специальной техники динамической генерации изображений на основе растровых шаблонов была обусловлена тем, что в настоящее время отсутствует единый общепринятый подход к отображению векторной графики в среде стандартного Интернет-браузера. Наиболее распространенный Интернет-браузер (Microsoft Internet Explorer) имеет развитые встроенные средства представления векторной графики — язык VML, который, однако, не является стандартом и поддерживается только компанией Microsoft. Консорциум W3C утвердил в качестве стандарта для представления векторной графики в Интернет язык SVG [3]. Однако SVG пока не входит в стандартную поставку современных Интернет-браузеров и требует специальной установки на компьютер клиента. Таким образом, использование разработанной техники динамической генерации изображений на основе растровых шаблонов — временная мера. Будущее, безусловно, за стандартизированными общедоступными средствами отображения векторной графики.

Для уменьшения объема информации, пересылаемой на компьютер клиента в начале сеанса работы WEB-приложения, были разработаны средства динамической подкачки программ и данных, а также средства синхронизации использования динамически подкачиваемых объектов. Это позволило ограничить начальный объем информации, необходимой для “раскрутки” WEB-приложения. Кроме того, для уменьшения объема передаваемой по сети информации были разработаны специальные средства упаковки, используемые для сжатия как программ, так и данных.

При создании дополнительного клиентского ПО был использован объектно-ориентированный подход. Это позволило реализовать открытую модульную архитектуру клиентского ПО, облегчающую модернизацию и наращивание функциональности клиентского ПО.

Все создаваемые в процессе работы WEB-приложения объекты данных можно разделить на следующие категории:

— содержательные данные, представление которых осуществляется WEB- приложением;

— метаданные, определяющие способы представления и параметры визуализации содержательных данных;

— метаданные, определяющие сценарии представления содержательных данных.

Содержательные данные, на представление которых ориентировано клиентское ПО, организованы в вектора данных с кодовой привязкой элементов. Каждый элемент такого вектора данных характеризуется принадлежностью к элементу некоторого множества, называемого кодовым пространством. Элементы одного вектора данных принадлежат элементам одного кодового пространства. Благодаря наличию связи элементов векторов данных с элементами кодового пространства, удается автоматически устанавливать соответствие между элементами различных векторов данных, принадлежащих одному кодовому пространству. Для установления соответствия между элементами векторов данных, принадлежащих различным кодовым пространствам, используется еще один вид объектов — кодовые проекции. Кодовые проекции задают соответствие между элементами различных кодовых пространств. Благодаря кодовым проекциями, удается автоматически устанавливать соответствие между элементами векторов данных, принадлежащих к различным кодовым пространствам.

Метаданные, определяющие способ визуального представления содержательных данных, организованы в объекты, называемые визуальными компонентами. Визуальные компоненты являются шаблонами представления содержательных данных. В форме визуальных компонент могут быть представлены не только визуальные представления данных, но и группы элементов управления. Визуальные компоненты отображаются в отдельных окнах. В терминах окон определяется состав одновременно видимой информации. Оконная модель лежит также в основе модели возбуждения и распространения программных событий, обеспечивающих согласованное изменение информации во множестве одновременно открытых окон в соответствии с информационными связями, существующими между отдельными фрагментами данных.

Метаданные, задающие параметры визуализации определенного способа представления содержательных данных, разделены на две группы: индивидуальные и общие. Индивидуальные параметры визуализации задаются при создании каждого экземпляра визуальной компоненты. Общие параметры сгруппированы в отдельные объекты, называемые стилями. Экземпляр стиля передается конструктору визуальной компоненты наряду с индивидуальными параметрами. Каждому типу визуальной компоненты соответствует свой тип стиля. Стиль содержит параметры визуального представления, такие как фон окна, атрибуты размещения элементов визуального представления, параметры шрифтов для текстовых элементов и т.д.

Существуют специальные классы объектов, участвующие в определении сценариев представления содержательных данных. К таким объектам можно отнести виртуальные окна, которые управляют со-

зданием физических окон браузера, в которых строится визуальное представление данных. Виртуальное окно определяет:

— будет ли создано новое физическое окно браузера или будет использовано уже существующее окно;

— будет ли это самостоятельное окно или фрейм в некотором окне браузера.

В случае создания нового физического окна браузера виртуальное окно задает:

— положение окна на экране монитора,

— начальный размер окна,

— возможность изменения размера окна,

— возможность прокрутки содержимого окна,

— признаки отображения стандартных элементов управления окна браузера (меню, панель инструментов, панель адреса).

Виртуальные окна организованы в специальные объекты “Схемы окон”. Эти объекты определяют стратегию воплощения визуальных компонент в окнах браузера.

К объектам, определяющим сценарий представления содержательных данных, следует отнести и объект “Группа визуальных компонент”. Группа задает список одновременно отображаемых визуальных компонент и соответствующий им список виртуальных окон, определяющих для каждой визуальной компоненты параметры порождения физического окна браузера.

Носителями элементов сценария являются и сами визуальные компоненты. Примером может служить визуальная компонента “Иерархическая структура”, предназначенная для создания визуального представления древовидной структуры, листовыми узлами которой могут быть визуальные компоненты или группы визуальных компонент. Если “кликнуть” мышкой на листовом узле иерархической структуры, то активизируется сценарий, который создает визуальное представление объекта, соответствующего данному узлу иерархической структуры.

Кроме того, визуальные компоненты могут содержать элементы управления, с которыми связаны сценарии, осуществляющие управление согласованным изменением внешнего вида и состава отображаемой информации.

Визуальная компонента любого конкретного типа является наследником базового класса визуальных компонент. Для этого базового класса реализованы все необходимые методы, обеспечивающие создание визуального воплощения объекта в окне браузера. Для разработки новой визуальной компоненты необходимо только переопределить абстрактные методы базового класса, генерирующие HTML-документ, обеспечивающий визуальное воплощение объекта.

Для удобства конструирования сложных элементов управления, входящих в состав визуальных компонент, была разработана библиотека элементов управления. Библиотека позволяет создавать такие элементы управления как иерархические меню, кнопки панели инструментов с несколькими состояниями и др.

2.2. Объектная модель. Все объекты клиентского ПО делятся на две категории — синхронные и асинхронные. Синхронные объекты полностью определяются в момент конструирования экземпляра объекта. Таким образом, использовать экземпляр синхронного объекта можно сразу же после завершения работы его конструктора.

Асинхронные объекты после завершения работы конструктора еще не готовы к использованию. Конструктор лишь создает базовый дескриптор объекта, который содержит минимальный набор полей, необходимый для организации подготовки объекта к реальному использованию. Для определения готовности асинхронного объекта служит специальный метод, который в случае необходимости и запускает процесс подготовки объекта к использованию. Асинхронные объекты предназначены для воплощения объектов, динамически подкачиваемых с сервера в процессе работы WEB-приложения. Такие объекты, как правило, представляют собой значительные по объему фрагменты содержательной публикуемой информации или параметров визуализации. Например, вектора данных или шаблоны картограмм территорий и др. С помощью асинхронных объектов воплощаются и составные объекты, использующие другие асинхронные объекты. Например, экземпляр визуальной компоненты “Картограмма” использует такие асинхронные объекты как вектор данных и шаблон картограммы. На рис. 1 и 2 показаны иерархии классов асинхронных и синхронных объектов.

2.3. Модель памяти. Все программы и объекты клиентского ПО размещаются в памяти некоторого главного управляющего окна. Главное управляющее окно может быть как самостоятельным окном браузера, так и фреймом, в том числе и невидимым. Для успешной работы WEB-приложения главное управляющее окно должно сохраняться в течение всего сеанса его работы. Закрытие главного управляющего окна вызывает закрытие всех окон браузера, открытых в процессе работы WEB-приложения.

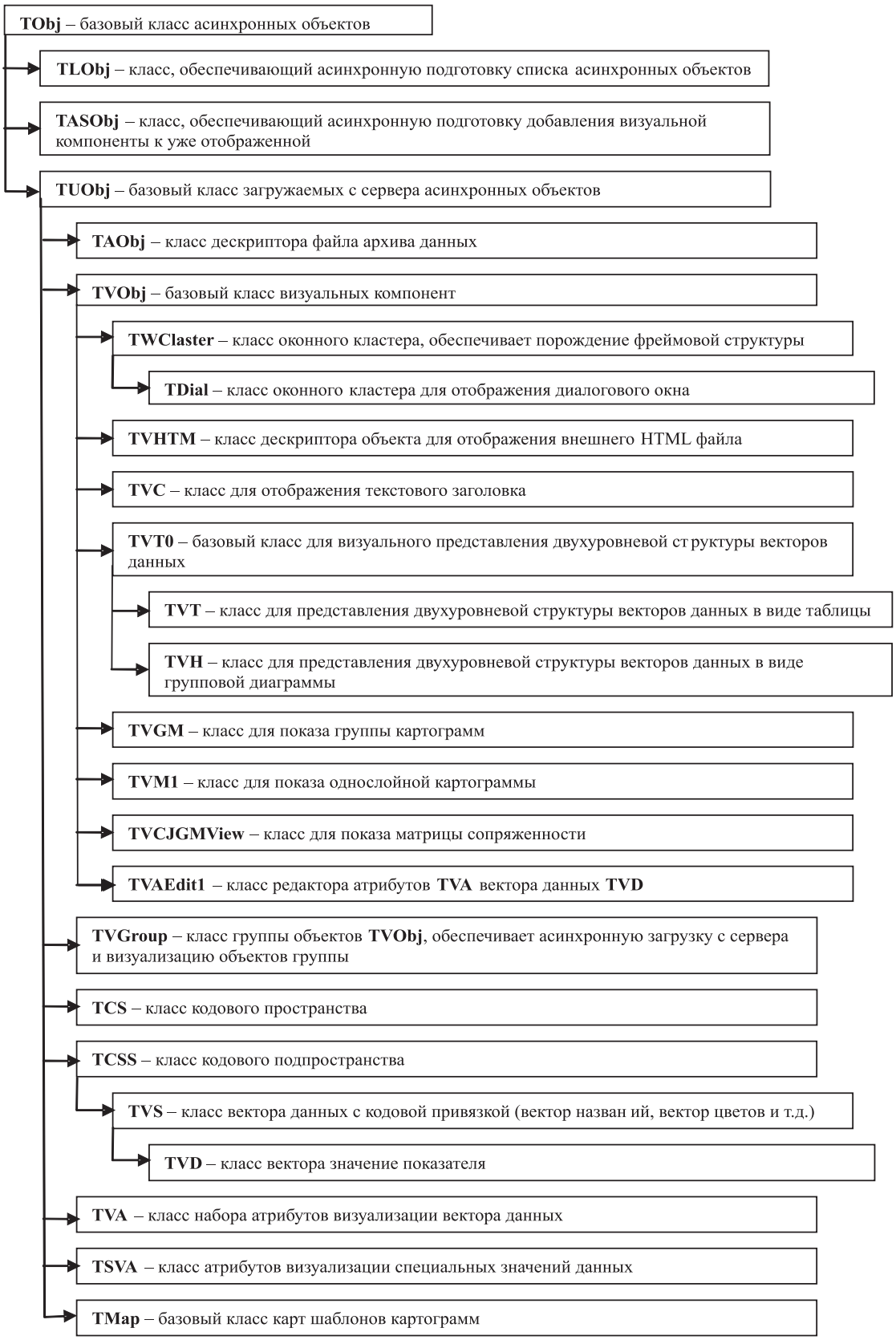


Рис. 1. Классы асинхронных объектов

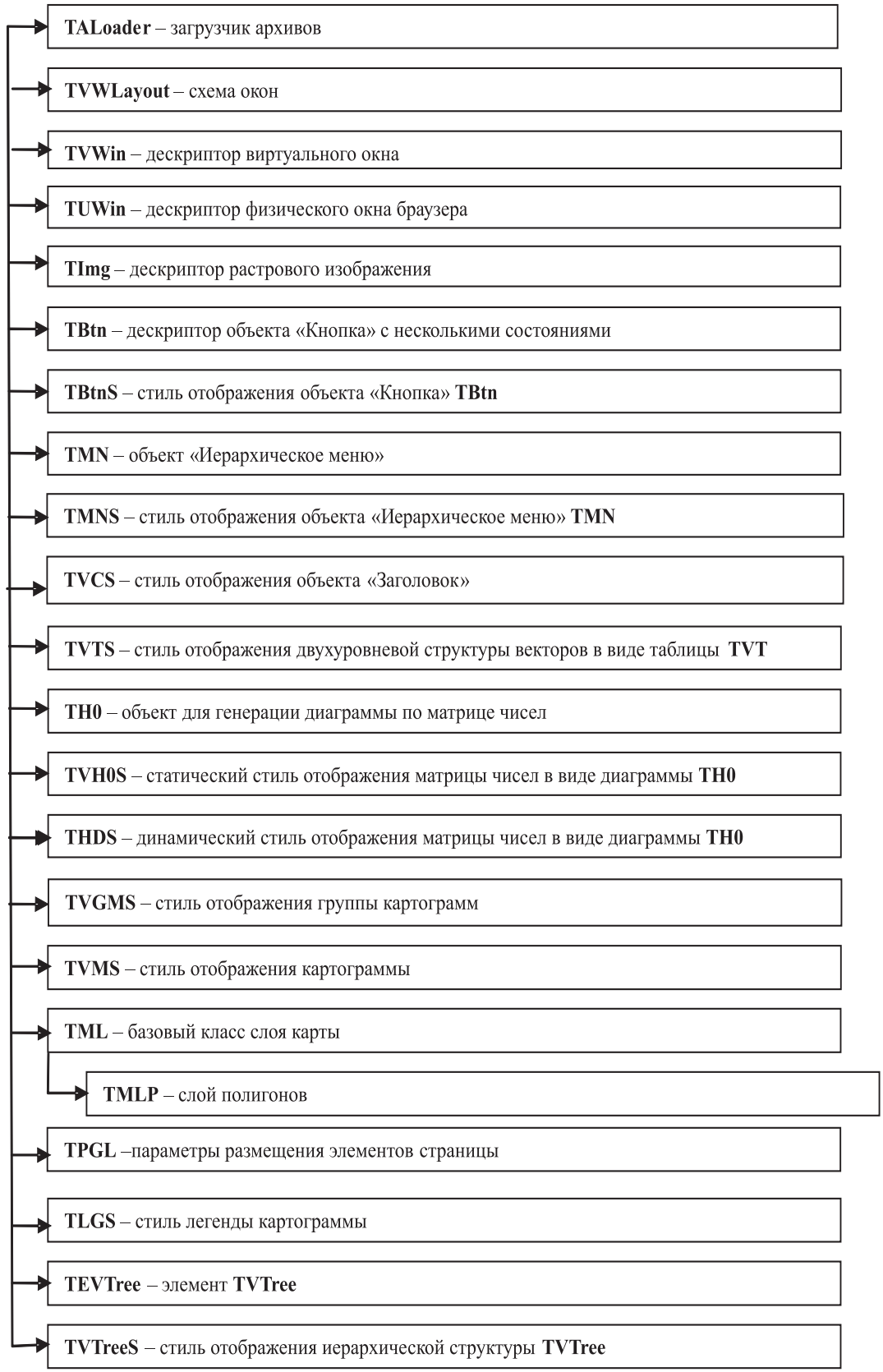


Рис. 2. Классы синхронных объектов

Объект window главного управляющего окна является контейнером для всех подкачиваемых с сервера программ и данных. В соответствии с принятыми соглашениями для обращения к объектам главного управляющего окна в самом главном управляющем окне и во всех вновь открываемых окнах создается глобальная переменная GW, являющаяся ссылкой на объект window главного управляющего окна. Таким образом, обращение к объектам, хранящимся в главном управляющем окне, всегда осуществляется через переменную GW — GW.Obj.

Данные некоторых типов размещаются в главном управляющем окне в специальных объектах, являющихся контейнерами. Причем в каждом контейнере накапливаются объекты определенного типа. Так, в контейнере GW.AU накапливаются подкачиваемые с сервера асинхронные объекты — наследники TUObj, а в контейнере GW.AW хранятся дескрипторы физических окон браузера TUWin.

2.4. Динамическая загрузка данных. Все программы и данные, которые должны быть загружены с сервера на компьютер клиента, размещаются в архивах — специальным образом организованных HTML-файлах. Загрузка архивов осуществляется объектом типа TALoader. Объект TALoader может осуществлять параллельную загрузку нескольких очередей архивов. Для организации процесса загрузки в главном управляющем окне создается несколько невидимых "загрузочных" фреймов. Каждому архиву при загрузке выделяется отдельный "загрузочный" фрейм. После завершения загрузки файла архива в "загрузочный" фрейм вызывается специальный метод объекта TALoader, который осуществляет перенос программ и данных из "загрузочного" фрейма в память главного управляющего окна.

Содержание архива представляет собой программный код обслуживания и порождения данных, оформленный в виде функций JavaScript (JS), код которых и переносится в главное управляющее окно. Соглашения на организацию архива предполагают, что список всех JS-функций, которые необходимо перенести в главное управляющее окно, должен в момент завершения загрузки размещаться в массиве SelfFuncs архива, а весь код порождения данных должен размещаться в функции с именем CreateLoadedData. После переноса кода JS-функций, которые перечислены в массиве SelfFuncs, из "загрузочного" фрейма в главное управляющее окно загрузчик осуществляет вызов перенесенной функции CreateLoadedData, которая и обеспечивает порождение загруженных данных в памяти главного управляющего окна.

2.5. Синхронизация объектов. Синхронизация готовности к использованию асинхронных объектов осуществляется следующим образом. Вначале с помощью конструктора создается базовый дескриптор асинхронного объекта, который имеет поле PrepList — список объектов, готовность которых нужно обеспечить до начала использования данного объекта, и поле WaitList — список объектов, ожидающих готовности данного объекта. Для подготовки асинхронного объекта A к использованию необходимо вызвать его метод Prepare

$$\text{A.Prepare(WObj)},$$

который возвращает код текущего состояния объекта A. Метод Prepare проверяет, начат ли процесс подготовки объекта A. Если объект A готов, то работа метода Prepare завершается, иначе объект WObj добавляется в список WaitList объекта A для последующего уведомления о завершении подготовки, и если процесс подготовки объекта A еще не начат, то метод Prepare осуществляет запуск процесса подготовки объектов из списка PrepList. Для этого выполняется вызов метода Prepare для каждого объекта из списка PrepList[0]:

$$\text{PrepList[0][i].Prepare(A)}.$$

Когда любой из этих объектов PrepList[0][i] достигает состояния готовности, он оповещает об этом объект A за счет вызова его метода Notify

$$\text{A.Notify(PrepList[0][i])}.$$

Когда все объекты из списка PrepList[0] достигнут состояния готовности, начинается работа со следующим элементом очереди до тех пор, пока не достигнут состояния готовности все объекты из очереди PrepList. Далее объект A выполняет собственную подготовку к использованию, после завершения которой он сам вызовет метод Notify для каждого объекта WaitList[i] из своего списка WaitList:

$$\text{WaitList[i].Notify(A)}.$$

Для использования любого объекта необходимо сначала обеспечить загрузку программ обслуживания объектов этого класса, затем — программ обслуживания для объектов из очереди PrepList, а потом — самих этих объектов. Синхронизация использования асинхронных объектов с загрузкой программ и данных с сервера осуществляется за счет того, что каждому файлу-архиву соответствует асинхронный объект TAObj, который может быть включен в очередь объектов PrepList любого асинхронного объекта. Метод

Prepare объекта TAObj вызывает начало загрузки соответствующего архива, а состояния готовности объект TAObj достигает после завершения загрузки архива.

Таким образом, если для работы некоторого асинхронного объекта A требуется определенный список объектов AList, то очередь PrepList его базового дескриптора должна иметь следующий состав. В первом элементе очереди должен быть список объектов TAObj архивов, которые содержат программы обслуживания объектов из списка AList. Во втором элементе очереди должен быть список объектов TAObj архивов, которые содержат код порождения базовых дескрипторов объектов из списка AList. В третьем элементе очереди должен размещаться сам список AList. И, наконец, в четвертом элементе очереди может содержаться список объектов TAObj архивов, которые содержат код окончательного определения объекта A до состояния готовности.

2.6. Схема визуализации. Схема визуализации основана на организации визуальных представлений данных в форме отображаемых в отдельных окнах браузера визуальных компонент. Один и тот же набор данных может быть визуализирован различным образом — в виде таблицы, диаграммы, картограммы, графика и т.д. Визуальная компонента определяет конкретный способ визуального представления выбранного набора данных. В ее задачу входит генерация HTML-документа, обеспечивающего соответствующее визуальное представление заданного набора данных. Этот HTML-документ может содержать помимо визуального представления данных элементы управления и программы их обслуживания. Элементы управления должны обеспечить пользователя возможностью в интерактивном режиме согласованно изменять не только данное визуальное представление, но и другие визуальные представления из некоторого одновременно отображаемого подмножества визуальных представлений. Базовым классом визуальных компонент является класс TVObj.

Для создания визуального представления некоторой визуальной компоненты VObj необходимо вызвать ее метод Show. Метод Show определяет дескриптор физического окна браузера UW (объект класса TUWin), в котором будет построено визуальное представление, и выполняет все необходимые действия по представлению визуальной компоненты в окне браузера. Процесс визуализации протекает по-разному в зависимости от того, в каком состоянии находится объект VObj и физическое окно браузера, соответствующее дескриптору физического окна UW.

Если объект VObj не готов к использованию (загружается сам или загружаются нужные ему объекты), то необходимо до начала визуализации дождаться его готовности. Это достигается за счет вызова его метода Prepare

$$\text{VObj.Prepare(UW)}.$$

Когда объект VObj достигнет готовности, будет вызван метод UW.Notify(VObj), который и завершит работу по визуализации:

— создаст физическое окно браузера;

— загрузит в это окно HTML-документ, текст которого генерирует метод MDoc объекта VObj;

— завершит формирования элементов основного HTML-документа с помощью метода Build объекта VObj после завершения загрузки основного документа.

Методы MDoc и Build индивидуальны для каждого класса визуальных компонент и определяют способ визуализации содержательных данных.

Если в момент вызова метода Show объект готов к использованию, то происходит анализ готовности физического окна браузера. Если окно полностью готово, т.е. открыто требуемое окно браузера и в нем уже загружен документ того же типа, что и документ вновь загружаемого визуального представления VObj, то сразу осуществляется вызов метода Build

$$\text{VObj.Build(UW)},$$

который переопределяет содержимое элементов основного HTML-документа в соответствии с визуализируемыми содержательными данными. В этом случае создание визуального представления выполняется наиболее быстро. Если же окно не готово, то вызывается метод Notify объекта UW

$$\text{UW.Notify(VObj)},$$

который выполняет перечисленные выше действия по завершению визуализации.

Следует особо рассмотреть случай генерации визуального представления объекта VObj во фрейме окна браузера, которое еще не создано. В этой ситуации визуализация осуществляется в несколько этапов. Дескриптор виртуального окна, предполагающего визуализацию во фрейме, содержит информацию о специальной визуальной компоненте WClaster (объект класса TWClaster), которая должна создать фреймовую структуру с нужным фреймом. В этом случае метод UW.Notify(VObj) осуществляет запуск процесса

визуализации визуальной компоненты WClaster за счет вызова его метода Show. Завершение визуализации объекта WClaster обеспечивает подготовку фрейма, предназначенного для размещения исходного объекта VObj. Готовность требуемого фрейма вновь активизирует метод соответствующего фрейму объекта UW

$$\text{UW.Notify(VObj)},$$

который выполняет перечисленные выше действия по визуализации данных исходного объекта VObj.

2.7. Управление окнами. Управление процессом выбора физического окна браузера при отображении визуальных представлений данных осуществляют объекты класса TVWLayout — "Схема окон". Объект "Схема окон" представляет собой набор виртуальных окон, которые используются для определения параметров физических окон браузера при отображении визуальных компонент. Виртуальное окно представляет собой совокупность параметров, определяющих свойства окна браузера, правила его создания и поведения после открытия. Правила создания определяют, будет ли использовано самостоятельное окно браузера или фрейм, уже существующее окно или новое и т.д. Правила поведения определяют размер и положения окна на экране монитора, возможность изменения его размеров и прокручивания его содержимого и т.д. Объект "Схема окон" и имя виртуального окна являются параметрами метода Show визуальной компоненты.

Каждому физическому окну браузера или фрейму соответствует экземпляр дескриптора физического окна TUWin. Объект TUWin хранит ссылку на окно или фрейм браузера, индикатор его состояния и состояния документа, загруженного в это окно, ссылку на отображенную в окне визуальную компоненту и т.д.

Для порождения в самостоятельном окне браузера структуры фреймов, которые могли бы использоваться для отображения визуальных компонент, служат объекты класса TWClaster — оконный кластер. Класс TWClaster является наследником класса TVObj. Головной документ объекта TWClaster должен содержать описание фреймовой структуры, которая может быть построена как на базе тегов <FRAME>, так и тегов <IFRAME>. Метод Build объекта TWClaster, вызываемый автоматически после завершения загрузки его основного документа, обеспечивает следующие действия:

— связывание элементов созданной фреймовой структуры с соответствующими дескрипторами физических окон;

— вызов метода Notify для каждого дескриптора фрейма, что в свою очередь обеспечивает продолжение загрузки визуальных компонент, ожидавших порождения фреймов.

Существует еще один класс объектов — TDial, обеспечивающий создание фреймовой структуры в окне браузера для отображения диалогов. Класс TDial является наследником класса TWClaster. Основной документ объекта TDial содержит два фрейма. Один — для показа объекта TVObj, определяющего содержание диалога, другой — для отображения набора кнопок ("OK", "Cancel" и д.р.), которые позволяют пользователю выбрать тип завершения диалога.

2.8. Модель событий. Модель событий, реализованная в клиентском ПО, позволяет управлять согласованным отображением информации в различных окнах в соответствии с информационными связями, существующими между отдельными фрагментами данных. Реакция визуальной компоненты на событие с именем Event оформлена как метод визуальной компоненты с тем же именем Event. Распространение событий осуществляется следующим образом. Сначала проводится анализ визуальных компонент всех открытых окон. Если визуальная компонента имеет соответствующий метод реакции на событие и окно, в котором она визуализирована, принадлежит к той же группе, что и окно отправителя события, то у этой визуальной компоненты вызывается метод реакции на событие с теми параметрами, которые были переданы источником события.

Метод визуальной компоненты, обеспечивающий реакцию на событие, имеет определенную структуру параметров. Первый параметр метода — имя дескриптора физического окна отправителя события. Последний параметр — дескриптор физического окна получателя события:

$$\text{Event(SenderUWName, EventSelfPar1, \ldots, EventSelfParN, AcceptorUW)}.$$

На остальные параметры метода (EventSelfPar1, ..., EventSelfParN) никаких ограничений не накладывается.

Возбуждение события осуществляется с помощью функции окна отправителя EventRaise. Эта функция выполняется на потоке окна отправителя и сохраняет параметры события в специальном буфере в этом окне. Затем она вызывает на потоке главного управляющего окна GW глобальную функцию EventTransp, которая осуществляет дальнейшее распространение события. Эта функция просматривает дескрипторы всех физических окон с целью поиска подходящих визуальных компонент получателей события, для которых она осуществляет вызов соответствующего метода Event с нужными параметрами.

Примером программного события, которое осуществляет согласованную смену информации в окнах, может служить событие “Смена текущего элемента кодового пространства”. Визуальная компонента “Таблица” при получении такого события выделяет элементы, соответствующие заданному элементу кодового пространства (при необходимости выполняется прокрутка окна с таблицей для того, чтобы выделенные элементы оказались в видимой области окна). Визуальная компонента “Групповая диаграмма” при получении такого события отображает диаграмму для соответствующих элементов данных. Визуальная компонента “Картограмма” при получении такого события наглядно выделяет элементы картограммы, соответствующие заданному в событии элементу кодового пространства.

2.9. Организация векторов данных. Данные, отображаемые с помощью визуальных компонент, организованы в вектора с кодовой привязкой. Каждый элемент такого вектора данных характеризуется принадлежностью к элементу некоторого множества, называемого кодовым пространством. Для обработки данных, проводимой при построении визуальных представлений, необходимо устанавливать соответствие между элементами таких векторов. Соответствие между элементами векторов устанавливается с помощью специальных связующих объектов — кодовых пространств, кодовых подпространств и кодовых проекций. Кодовые пространства представляют собой множество элементов, характеризующихся определенным уникальным кодом. Соответствие элементов вектора данных с кодовой привязкой и элементов кодового пространства устанавливается с помощью кодового подпространства. Кодовое подпространство представляет собой упорядоченное подмножество элементов кодового пространства. Оно реализовано как массив индексов элементов кодового пространства, синхронный с вектором данных, и, кроме того, кодовое подпространство имеет ссылку на свое кодовое пространство. Каждый вектор данных с кодовой привязкой имеет ссылку на соответствующее ему кодовое подпространство. Вектора данных, имеющие одинаковую структуру элементов, ссылаются на одно и то же кодовое подпространство.

Установление соответствия между элементами кодовых пространств обеспечивается кодовыми проекциями. Кодовую проекцию можно представить как однозначную функцию, областью определения которой являются множество всех элементов одного кодового пространства, а областью значений — подмножество элементов другого кодового пространства. Кодовая проекция реализована как вектор с кодовой привязкой, который синхронен элементам кодового пространства, являющегося ее областью определения. Значениями элементов этого вектора служат индексы элементов кодового пространства, которое задает область значений кодовой проекции. Таким образом, кодовая проекция является одновременно не только вектором с кодовой привязкой, но и кодовым подпространством кодового пространства своей области значений.

Для установления соответствия между элементами векторов данных с кодовой привязкой достаточно задать соответствие между элементами их кодовых подпространств. Такое соответствие называется проекцией данных. Проекцию данных, как и кодовую проекцию, можно представить как однозначную функцию, областью определения которой является множество всех элементов одного вектора данных, а областью значений — подмножество элементов другого вектора данных. Если кодовые подпространства принадлежат различным кодовым пространствам, то для построения проекции данных необходимо использовать проекцию соответствующих кодовых пространств. Проекция данных реализована как массив, который синхронен элементам вектора данных, являющегося ее областью определения. Значениями элементов этого массива служат индексы элементов вектора данных, который задает область значений проекции данных.

2.10. Визуальные компоненты для отображения групп векторов. Для отображения иерархических структур векторов данных создан ряд классов визуальных компонент, наследников TVObj. Эти визуальные компоненты позволяют отображать двухуровневую структуру, состоящую из нескольких одноуровневых секций векторов данных в виде таблицы, в виде групповой линейчатой диаграммы и в виде групповой картограммы. Комбинирование различных видов визуальных представлений одной и той же структуры исходных данных обеспечивает ее комплексное отображение и анализ.

2.10.1. Таблица. Табличный вид (рис. 3) позволяет наиболее полно представить двухуровневую структуру векторов данных. Вектора данных отображаются как столбцы таблицы. Столбцы одной секции имеют дополнительный общий заголовок. Табличный вид содержит элементы управления, позволяющие динамически изменять внешний вид табличного представления:

— управлять видимостью заголовков столбцов и секций,

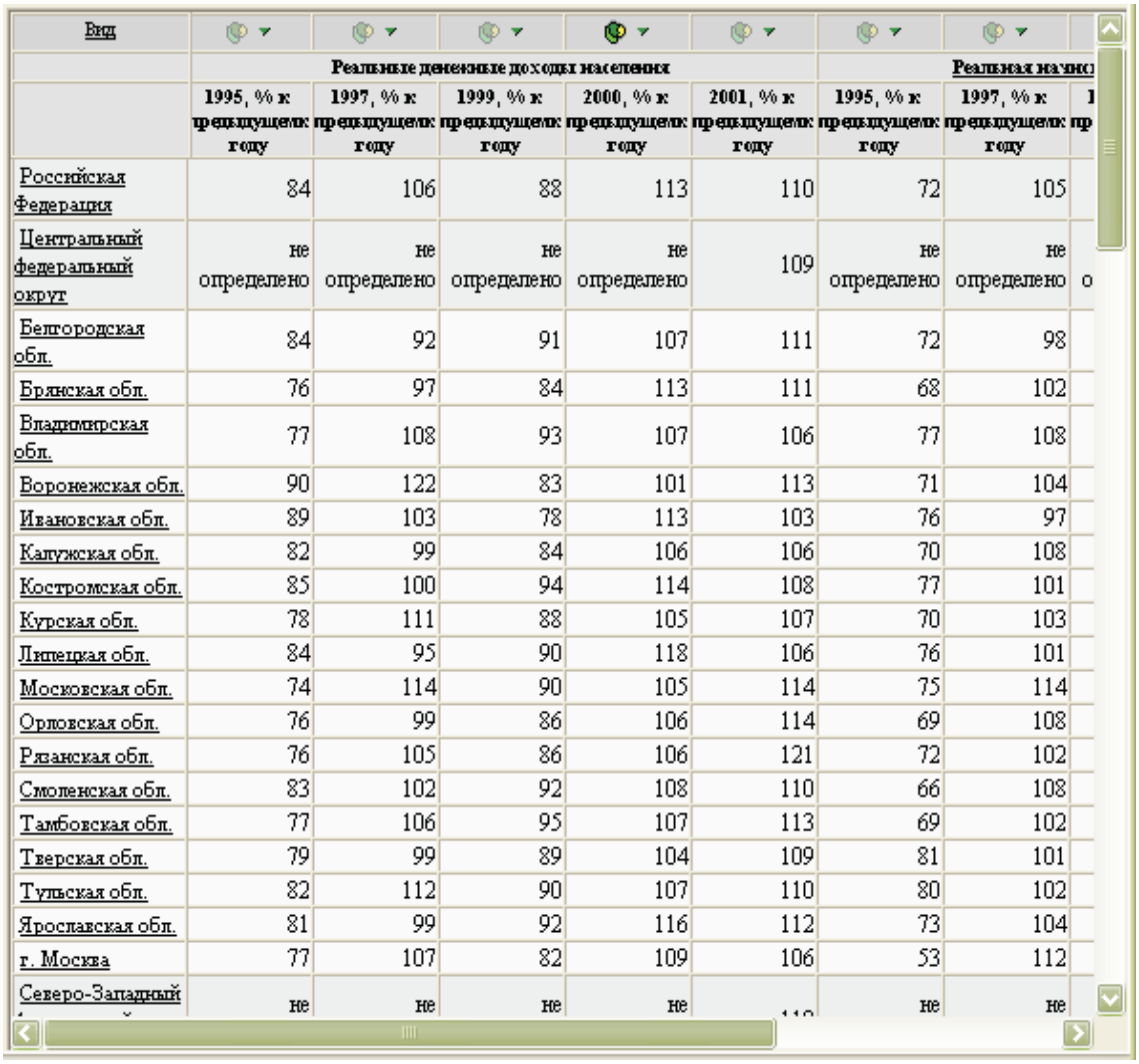
— изменять порядок строк и столбцов вручную,

— изменять состав визуализируемых векторов,

— сортировать строки по выбранным векторам данных,

— сортировать столбцы по выбранным строкам,

— закрашивать ячейки таблицы в соответствии с установленными параметрами разбиения множества



Вид	Реальные денежные доходы населения					Реальная начис		
	1995, % к предыдущему году	1997, % к предыдущему году	1999, % к предыдущему году	2000, % к предыдущему году	2001, % к предыдущему году	1995, % к предыдущему году	1997, % к предыдущему году	пр
Российская Федерация	84	106	88	113	110	72	105	
Центральный федеральный округ	не определено	не определено	не определено	не определено	109	не определено	не определено	о
Белгородская обл.	84	92	91	107	111	72	98	
Брянская обл.	76	97	84	113	111	68	102	
Владимирская обл.	77	108	93	107	106	77	108	
Воронежская обл.	90	122	83	101	113	71	104	
Ивановская обл.	89	103	78	113	103	76	97	
Калужская обл.	82	99	84	106	106	70	108	
Костромская обл.	85	100	94	114	108	77	101	
Курская обл.	78	111	88	105	107	70	103	
Липецкая обл.	84	95	90	118	106	76	101	
Московская обл.	74	114	90	105	114	75	114	
Орловская обл.	76	99	86	106	114	69	108	
Рязанская обл.	76	105	86	106	121	72	102	
Смоленская обл.	83	102	92	108	110	66	108	
Тамбовская обл.	77	106	95	107	113	69	102	
Тверская обл.	79	99	89	104	109	81	101	
Тульская обл.	82	112	90	107	110	80	102	
Ярославская обл.	81	99	92	116	112	73	104	
г. Москва	77	107	82	109	106	53	112	
Северо-Западный	не	не	не	не		не	не	

Рис. 3

значений элементов вектора на классы,

— изменять параметры разбиения множества значений элементов вектора на классы,

— отображать рядом с ячейкой с числовым значением столбик линейчатой диаграммы, соответствующий этому числовому значению,

— транспонировать таблицу.

Кроме того, табличное представление содержит элементы управления, которые позволяют согласованно изменять содержимое других одновременно отображаемых визуальных компонент:

— влиять на текущую картограмму в групповой картограмме и на состав элементов в групповой диаграмме через изменение текущего вектора данных,

— влиять на состав элементов в групповой диаграмме через изменение текущего элемента векторов данных.

2.10.2. Групповая диаграмма. Групповая диаграмма (рис. 4) позволяет в каждый момент времени отобразить в виде линейчатой диаграммы матрицу чисел, составленную из некоторого подмножества элементов всех векторов одной секции многосекционной структуры данных. Групповая диаграмма дает возможность изменять в динамике подмножество отображаемых данных. Эти изменения могут происходить как за счет воздействия внешних событий, так и за счет активизации элементов управления самой групповой диаграммы. Групповая диаграмма содержит элементы управления, позволяющие менять ее вид:

— изменять состав отображаемой информации за счет выбора секции,

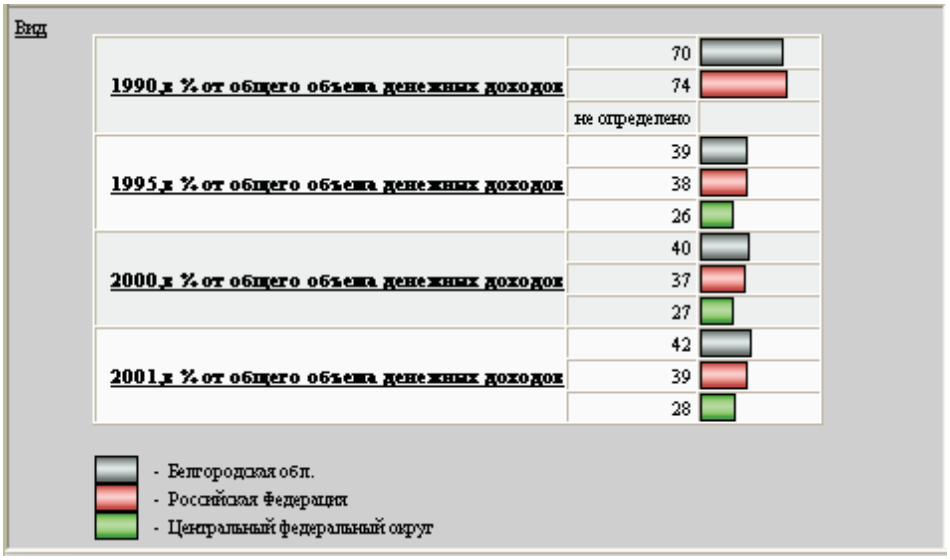


Рис. 4

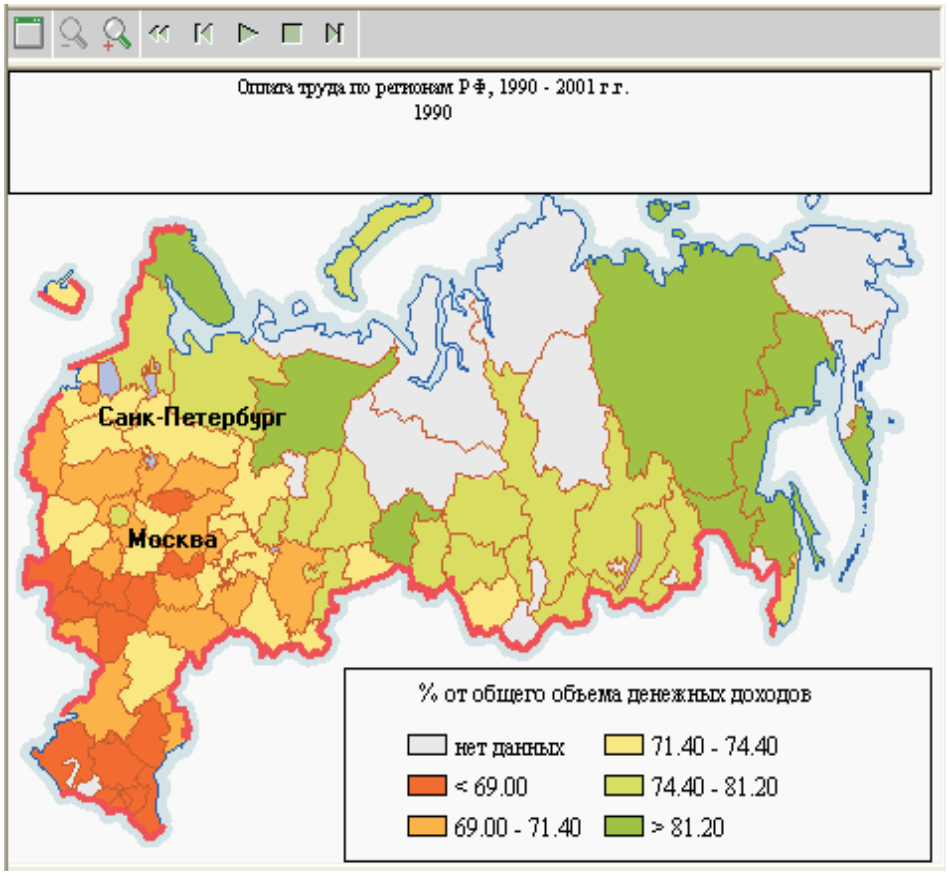


Рис. 5

— изменять группировку элементов матрицы,
— изменять размещение названий групп и элементов,
— изменять направление размещения групп,
— отображать элементы одной группы в виде составной диаграммы.

2.10.3. Групповая картограмма. Групповая картограмма (рис. 5) обеспечивает отображение век-

торов многосекционной структуры данных в виде картограммы. Сам объект “Групповая картограмма” отображается в отдельном окне в виде совокупности элементов управления, которые позволяют:

— последовательно в прямом и обратном порядке просматривать картограммы, построенные по всем векторам, входящим в многосекционную структуру векторов данных;

— запускать режим “плавного” последовательного воспроизведения всех картограмм для векторов одной секции;

— изменять масштаб изображения картограмм.

2.11. Растровые шаблоны картограмм. Для построения динамических изображений была разработана техника динамической генерации в среде стандартного Интернет-браузера параметризованных содержательными данными изображений на основе растровых шаблонов [4]. Разработанная техника не зависит от программно-аппаратной платформы клиента и использует только средства HTML и JavaScript без привлечения дополнительных программных надстроек (Java, Flash, ActiveX и т.д.). Она состоит в комбинировании растровых изображений и динамически сгенерированных абсолютно позиционированных элементов <DIV> с заданным цветом фона. Элементы <DIV> задают области изображения, цвет которых должен изменяться динамически в зависимости от значения данных. Статическая часть изображения формируется с помощью заранее подготовленных растров.

Эта техника была использована для динамического построения картограмм по векторам числовых данных. Результирующее изображение картограммы строится на основании шаблона картограммы, состоящего из соответствующего растрового изображения, и описания координат прямоугольных областей и их принадлежности к определенным объектам территории. Построение картограммы по вектору данных, каждый элемент которого соответствует некоторому объекту территории, осуществляется следующим образом. Вначале для каждого элемента вектора определяется цвет, которым соответствующий элемент будет отображаться в картограмме. Затем, в соответствии с заданными в шаблоне координатами и принадлежностью к объектам территории, осуществляется генерация HTML-документа, который содержит цветные, абсолютно позиционированные элементы <DIV> и размещенное поверх них статическое растровое изображение, которое имеет прозрачные пиксели, в тех областях, цвет которых должен изменяться динамически.

Шаблон изображения картограммы (координаты прямоугольных элементов и статическое растровое изображение) один раз передается на компьютер клиента. Затем он многократно используется для динамического построения картограмм по векторам данных. Таким образом, для построения новой картограммы на клиентский компьютер необходимо передать лишь новый вектор числовых данных.

Шаблон картограммы обеспечивает генерацию не масштабируемого изображения, поэтому объект “Картограмма” для построения изображений различного масштаба должен использовать различные шаблоны картограмм. Шаблоны картограмм различных масштабов для одной территории могут отличаться не только размером генерируемого изображения, но и составом отображаемой на них информации.

Реализованные в клиентском ПО средства задания шаблонов картограмм позволяют описывать изображения, содержащие несколько тематических слоев, параметризованных данными. Каждый параметризованный данными слой изображения может иметь собственную легенду. Для вывода легенды каждого слоя определяется самостоятельная прямоугольная область на странице с картограммой. Страница с картограммой состоит из нескольких самостоятельных прямоугольных областей:

— заголовок картограммы,

— изображение картограммы,

— легенды тематических слоев.

Для создания изображения в каждой области страницы генерируется собственный HTML-документ.

Параметризованный данными тематический слой изображения карты является интерактивным. При попадании курсора в область картограммы, связанную с элементом вектора данных, который формировал параметры отображения этой области, появляется название соответствующего элемента вектора и его значение. Нажатие кнопки “мыши” над областью картограммы вызывает событие — “Сменился текущий элемент кодового пространства”. Параметры этого события, имя кодового пространства и индекс его элемента формируются по элементу вектора данных, который соответствует той области изображения карты, над которой находится курсор.

3. Заключение. Получены следующие основные результаты:

— разработана программная модель клиентского ПО и соглашения на организацию файлов для хранения данных и программ, позволившие создать основу для построения ряда конструкторов электронных публикаций для широкого класса задач представления данных, в том числе для представления данных, имеющих привязку к объектам территории;

— разработана открытая объектная модель клиентского ПО, ориентированная на развитие имеющихся и создание новых классов визуальных компонент;

— разработана независящая от программно-аппаратной платформы техника динамической генерации в среде стандартного Интернет-браузера параметризованных содержательными данными изображений на основе растровых шаблонов, которые используют только средства HTML и JavaScript без привлечения специализированных программных надстроек (Java, Flash, ActiveX и т.д.);

— разработана библиотека типовых элементов управления (иерархическое меню, кнопка панели инструментов и т.д.), облегчающая создание визуальных компонент;

— разработаны программные средства сжатия программ и данных для пересылки по сети, позволяющие значительно уменьшить объем информации, которую необходимо передать на компьютер клиента.

СПИСОК ЛИТЕРАТУРЫ

1. *Богомолов Н.А., Ковалев А.Д., Синицын М.Н.* Об одном подходе к отображению векторной графической информации в среде Интернет-браузера // Вычислительные методы и программирование. 2002. **3**, № 2. 151–157.
2. *Богомолов Н.А., Ковалев А.Д., Синицын М.Н.* Методика отображения территориально привязанной информации в Интернет // Телематика'2002: Труды Всероссийской научно-методической конференции. СПб.: Изд-во Санкт- Петербургского технического университета, 2002. 96–97.
3. *Богомолов Н.А., Ковалев А.Д., Синицын М.Н.* Использование стандарта SVG для визуализации данных в Интернет // Научный сервис в сети Интернет: Труды Всероссийской научной конференции. 2003. 193–195.
4. *Арушанян О.Б., Богомолов Н.А., Ковалев А.Д., Синицын М.Н.* Об одном подходе к построению графических изображений в среде Интернет-браузера // Научный сервис в сети Интернет: Труды Всероссийской научной конференции. М.: Изд-во МГУ, 2004. 132–133.
5. ECMAScript Language Specification, Standard ECMA-262, 3rd edition (December 1999).
6. *Гудман Д.* JavaScript — Библия пользователя, 4-е издание / Пер. с англ. М.: Издательский дом "Вильямс", 2002.

Поступила в редакцию
30.09.2004