C++Builder и современные информационные технологии

C++Builder — одна из самых мощных систем, позволяющих на самом современном уровне создавать как отдельные прикладные программы Windows, так и разветвленные комплексы, предназначенные для работы в корпоративных сетях и в Интернет. Но прежде, чем рассказывать о C++Builder, надо сделать краткий обзор тех современных информационных технологий, которые он поддерживает.

Объектно-ориентированное программирование

Объектно-ориентированное программирование (сокращенно ООП) — это в наше время совершенно естественный подход к построению сложных (и не очень сложных) программ и систем. Когда вы открываете любую программу Windows, вы видите окно с множеством кнопок, разделов меню, окон редактирования, списков и т.п. Все это *объекты*. Причем сами по себе они ничего не делают. Они ждут каких-то *событий* — нажатия пользователем клавиш или кнопок мыши, перемещения курсора и т.д. Когда происходит подобное событие, объект получает *сообщение* об этом и как-то на него реагирует: выполняет некоторые вычисления, разворачивает список, заносит символ в окно редактирования. Вот такая программа Windows и есть объектно-ориентированная программа (для краткости будем называть прикладные программы *приложениями*).

Приложение, построенное по принципам объектной ориентации — это не последовательность каких-то операторов, не некий жесткий алгоритм. Объектно-ориентированная программа — это совокупность объектов и способов их взаимодействия. Отдельным (и главным) объектом при таком подходе во многих случаях можно считать пользователя программы. Он же служит и основным, но не единственным, источником событий, управляющих приложением.

Попробуем разобраться с основным понятием ООП — объектом. Для начала можно определить объект как некую *совокупность данных и способов работы с ними*. Данные можно рассматривать как поля записи. Это характеристики объекта. Пользователь и объекты программы должны, конечно, иметь возможность читать эти данные объекта, как-то их обрабатывать и записывать в объект новые значения.

Здесь важнейшее значение имеют *принципы инкапсуляции и сокрытия данных*. Принцип сокрытия данных заключается в том, что внешним объектам и пользователю прямой доступ к данным, как правило, запрещен. Делается это из двух соображений.

Во-первых, для надежного функционирования объекта надо поддерживать *целостность и непротиворечивость* его данных. Если не позаботиться об этом, то внешний объект или пользователь могут занести в объект такие неверные данные, что он начнет функционировать с ошибками.

Во-вторых, необходимо изолировать внешние объекты от особенностей внутренней реализации данных. Для внешних потребителей данных должен быть доступен только *пользовательский интерфейс* — описание того, какие имеются данные и функции и как их использовать. А внутренняя реализация — это дело разработчика объекта. При таком подходе разработчик может в любой момент модернизировать объект, изменить структуру хранения и форму представления данных, но, если при этом не затронут интерфейс, внешний потребитель этого даже не заметит. И, значит, во внешней программе и в поведении пользователя ничего не придется менять.

Чтобы выдержать принцип сокрытия данных, в объекте обычно определяются процедуры и функции, обеспечивающие все необходимые операции с данными: их чтение, преобразование, запись. Эти функции и процедуры называются *методами* и через них происходит общение с данными объекта.

Совокупность данных и методов их чтения и записи называется *свойством*. Свойства можно устанавливать в процессе проектирования, их можно изменять программно во время выполнения вашей прикладной программы. Причем внешне это все

выглядит так, как будто объект имеет какие-то данные, например, целые числа, которые можно прочитать, использовать в каких-то вычислениях, заложить в объект новые значения данных. В процессе проектирования вашего приложения с помощью C++ Builder вы можете видеть значения некоторых из этих данных в окне Инспектора Объектов, можете изменять эти значения. В действительности все обстоит иначе. Все общение с данными происходит через методы их чтения и записи. Это происходит и в процессе проектирования, когда среда C++Builder запускает в нужный момент эти методы, и в процессе выполнения приложения, поскольку компилятор C++Builder незримо для разработчика вставляет в нужных местах программы вызовы этих методов.

Помимо методов, работающих с отдельными данными, в объекте имеются методы, работающие со всей их совокупностью, меняющие их структуру. Таким образом, объект является *совокупностью свойств и методов*. Но это пока нельзя считать законченным определением объекта, поскольку прежде, чем дать полное определение, надо еще рассмотреть взаимодействие объектов друг с другом.

Средой взаимодействия объектов (как бы силовым полем, в котором существуют объекты) являются *сообщения*, генерируемые в результате различных событий. События наступают, прежде всего, вследствие действий пользователя — перемещения курсора мыши, нажатия кнопок мыши или клавиш клавиатуры. Но события могут наступать и в результате работы самих объектов. В каждом объекте определено множество событий, на которые он может реагировать. В конкретных экземплярах объекта могут быть определены *обработчики* каких-то из этих событий, которые и определяют реакцию данного экземпляра объекта. К написанию этих обработчиков, часто весьма простых, и сводится, как будет видно далее, основное программирование при разработке *графического интерфейса пользователя* с помощью C++Builder.

Теперь можно окончательно определить объект как *совокупность свойств и методов, а также событий, на которые он может реагировать*. Условно это изображено на рисунке 1. Внешнее управление объектом осуществляется через обработчики событий. Эти обработчики обращаются к методам и свойствам объекта. Начальные значения данных объекта могут задаваться также в процессе проектирования установкой различных свойств. В результате выполнения методов объекта могут генерироваться новые события, воспринимаемые другими объектами программы или пользователем.

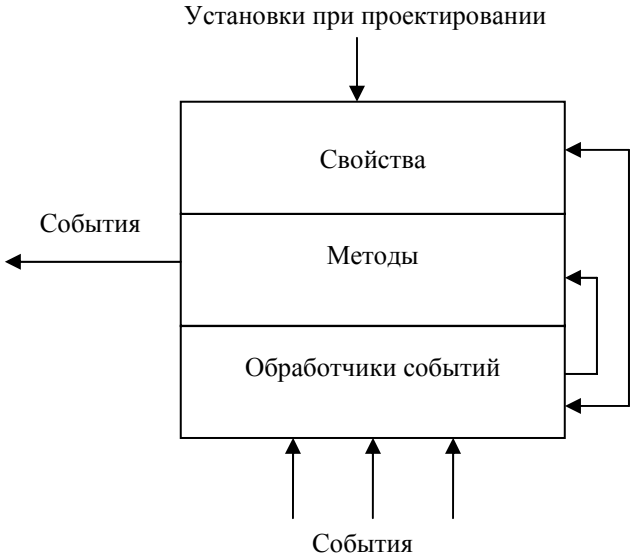

Рисунок 1. Схема организации объекта.

Представление о программе как о некоторой фиксированной совокупности

объектов не является полным. Чаще всего сложная программа — это не просто какая-то предопределенная совокупность объектов. В процессе работы объекты могут создаваться и уничтожаться. Таким образом, структура программы является *динамическим образованием*, меняющимся в процессе выполнения. Основная цель создания и уничтожения объектов — экономия ресурсов компьютера и, прежде всего, памяти. Несмотря на бурное развитие вычислительной техники, память, наверное, всегда будет лимитировать возможности сложных приложений. Это связано с тем, что сложность программных проектов растет теми же, если не более быстрыми, темпами, что и техническое обеспечение. Поэтому от объектов, которые не нужны на данной стадии выполнения программы, нужно освобождаться. При этом освобождаются и выделенные им области памяти, которые могут использоваться вновь создаваемыми объектами. Простой пример этого — окно-заставка с логотипом, появляющееся при запуске многих приложений. После начала реального выполнения приложения эта заставка исчезает с экрана и никогда больше не появится в данном сеансе работы. Было бы варварством не уничтожить этот объект и не освободить занимаемую им память для более продуктивного использования.

С целью организации динамического распределения памяти во все объекты заложены методы их создания — *конструкторы* и уничтожения — *деструкторы*. Конструкторы объектов, которые изначально должны присутствовать в приложении (прикладной программе), срабатывают при запуске программы. Деструкторы всех объектов, имеющихся в данный момент в приложении, срабатывают при завершении его работы. Но нередко и в процессе выполнения различные новые объекты (например, новые окна документов) динамически создаются и уничтожаются с помощью их конструкторов и деструкторов.

Включать объекты в свою программу можно двумя способами: *вручную* включать в нее соответствующие операторы (это приходится делать не очень часто) или путем визуального программирования, используя заготовки — *компоненты*.

Основы визуального программирования интерфейса

Сколько существует программирование, столько существуют в нем и тупики, в которые оно постоянно попадает и из которых, в конце концов, доблестно выходит. Один из таких тупиков или кризисов не так давно был связан с разработкой графического интерфейса пользователя. Программирование вручную всяких привычных пользователю окон, кнопок, меню, обработка событий мыши и клавиатуры, включение в программы изображений и звука требовало все больше и больше времени программиста. В ряде случаев весь этот сервис начинал занимать до 80—90% объема программных кодов. Причем весь этот труд нередко пропадал почти впустую, поскольку через год — другой менялся общепринятый стиль графического интерфейса и все приходилось начинать заново.

Выход из этой ситуации обозначился благодаря двум подходам. Первый из них — стандартизация системных функций и появление пользовательских интерфейсов API. В них описаны функции, переменные, константы, к которым разработчик может обращаться из своего приложения. Благодаря принципу скрытия данных, пользовательское приложение перестало зависеть от реализации тех или иных функций. В итоге при смене стиля графического интерфейса (например, при переходе от Windows 3.x к Windows 95, или от Windows 98 к Windows XP) приложения смогли автоматически приспосабливаться к новой системе без какого-либо перепрограммирования. На этом пути создались прекрасные условия для решения одной из важнейших задач совершенствования техники программирования — *повторного использования кодов*. Однажды разработанные вами формы, компоненты, функции могли быть впоследствии неоднократно использованы вами или другими программистами для решения их задач. Каждый программист получил доступ к наработкам других программистов и к огромным библиотекам, созданным

различными фирмами. Причем была обеспечена совместимость программного обеспечения, разработанного на разных алгоритмических языках.

Вторым революционным шагом, кардинально облегчившим жизнь программистов, явилось появление визуального программирования, возникшего в Visual Basic и нашедшего блестящее воплощение в системах C++Builder и Delphi фирмы Borland. Это явилось решающим шагом в развитии так называемой CASE-технологии (Computer Aided Software Engineering — автоматизированное проектирование программного обеспечения).

Визуальное программирование позволило свести проектирование пользовательского интерфейса к простым и наглядным процедурам, которые дают возможность за минуты или часы сделать то, на что ранее уходили месяцы работы. В современном виде в C++Builder это выглядит так.

Вы работаете в Интегрированной Среде Разработки (ИСР или Integrated development environment — IDE) C++ Builder. Среда предоставляет вам *формы* (в приложении их может быть несколько), на которых размещаются компоненты. Обычно это оконные формы, хотя они могут быть сделаны невидимыми. На форму с помощью мыши переносятся и размещаются пиктограммы компонентов, имеющихся в библиотеках C++Builder. С помощью простых манипуляций вы можете изменять размеры и расположение этих компонентов. При этом вы все время в процессе проектирования видите результат — изображение формы и расположенных на ней компонентов. Вам не надо мучиться, многократно запуская приложение и выбирая наиболее удачные размеры окна и компонентов. Результаты проектирования вы видите, даже не компилируя программу, немедленно после выполнения какой-то операции с помощью мыши.

Но достоинства визуального программирования не сводятся к этому. Самое главное заключается в том, что во время проектирования формы и размещения на ней компонентов C++Builder автоматически формирует коды программы, включая в нее соответствующие фрагменты, описывающие данный компонент. А затем в соответствующих диалоговых окнах пользователь может изменить заданные по умолчанию значения каких-то свойств этих компонентов и, при необходимости, написать обработчики каких-то событий. То есть проектирование сводится, фактически, к размещению компонентов на форме, заданию некоторых их свойств и написанию, при необходимости, обработчиков событий.

Компоненты могут быть *визуальные*, видимые при работе приложения, и *невизуальные*, выполняющие те или иные служебные функции. Визуальные компоненты сразу видны на экране в процессе проектирования в таком же виде, в каком их увидит пользователь во время выполнения приложения. Это позволяет очень легко выбрать место их расположения и их дизайн — форму, размер, оформление, текст, цвет и т.д. Невизуальные компоненты видны на форме в процессе проектирования в виде пиктограмм, но пользователю во время выполнения они не видны, хотя и выполняют для него за кадром весьма полезную работу.

В библиотеки визуальных компонентов C++ Builder включено множество типов компонентов, и их номенклатура очень быстро расширяется от версии к версии. Имеющегося уже сейчас вполне достаточно, чтобы построить практически любое самое замысловатое приложение, не прибегая к созданию новых компонентов. При этом даже неопытный программист, делающий свои первые шаги на этом поприще, может создавать приложения, которые выглядят совершенно профессионально.

Типы объектов и, в частности, компонентов библиотек C++Builder оформляются в виде *классов*. Классы — это типы, определяемые пользователем. В классах описываются свойства объекта, его методы и события, на которые он может реагировать. Язык программирования C++ предусматривает только инструментарий создания классов. А сами классы создаются разработчиками программного обеспечения. Впрочем, создатели C++Builder уже разработали для вас множество очень полезных классов и включили их в библиотеки системы. Этими классами вы пользуетесь при работе в Интегрированной

Среде Проектирования. Конечно, это нисколько не мешает создавать вам свои новые классы.

Если бы при создании нового класса вам пришлось все начинать с нуля, то эффективность этого занятия была бы под большим сомнением. Да и разработчики C++Builder вряд ли создали бы в этом случае такое множество классов. Действительно, представьте себе, что при разработке нового компонента, например, какой-нибудь новой кнопки, вам пришлась бы создавать все: рисовать ее изображение, описывать все свойства, определяющие ее место расположения, размеры, надписи и картинки на ее поверхности, цвет, шрифты, описывать методы, реализующие ее поведение — изменение размеров, видимость, реакции на сообщения, поступающие от клавиатуры и мыши. Вероятно, представив себе все это, вы отказались бы от разработки новой кнопки.

К счастью, в действительности все обстоит гораздо проще, благодаря одному важному свойству классов — *наследованию*. Новый класс может наследовать свойства, методы, события своего *родительского класса*, т.е. того класса, на основе которого он создается. Например, при создании новой кнопки можно взять за основу один из уже разработанных классов кнопок и только добавить к нему какие-то новые свойства или отменить какие-то свойства и методы родительского класса.

Благодаря визуальному объектно-ориентированному программированию была создана технология, получившая название *быстрая разработка приложений*, по-английски RAD — Rapid Application Development. Эта технология характерна для нового поколения систем программирования, к которому относится и C++Builder.

Вы увидите, что C++Builder действительно позволяет очень быстро разрабатывать прикладные программы самого разного назначения и, прежде всего — программы для работы с базами данных. В этой области C++Builder занимает самые передовые позиции, работая с любыми системами управления базами данных.

В целом C++Builder — великолепный инструмент, как для начинающих программистов, так и для ассов программирования. Так что не зря вы заинтересовались C++Builder и, надеюсь, с пользой ознакомитесь с данной книгой.

Взаимодействие приложений в информационных системах

ООП и визуальное проектирование позволяют создавать прекрасные прикладные программы. Но в настоящее время приложения, разрабатываемые для различных предприятий и их подразделений, как правило, должны функционировать не сами по себе, а являться частью некоторой информационной системы. В этом случае один из основных вопросов — организация взаимного общения приложений друг с другом и с хранилищами информации — *базами данных* (БД).

Как правило, приложения, работающие в составе информационной системы, черпают информацию из баз данных, к которым имеют доступ и другие приложения. При этом естественным образом создается возможность общения приложений через данные. Например, одно приложение может записать результаты своей работы в БД, а другое — прочитать эти данные и использовать их в своей работе. Такое простейшее общение требует только одного — унификации данных, форматов их хранения и языка запросов к БД. Последнее решается, чаще всего, с помощью языка SQL.

Во многих случаях подобного общения через данные недостаточно для эффективной работы системы. Приложение-потребитель данных не может ждать, пока кто-то запустит приложение, поставляющее эти данные. Значит, нужна возможность запускать из одного приложения другие, передавая в них какую-то информацию. Запуск внешнего приложения называется порождением *процесса*. Дочерний процесс может выполняться в адресном пространстве родительского процесса, т.е. как бы встраиваться в породивший его процесс, а может выполняться в другом, собственном адресном пространстве и в другом параллельном *потоке*.

Запуск из одного приложения других приложений позволяет использовать

результат работы дочернего процесса, но и только. Часто этого мало. Требуется обмен информацией между приложениями, выполняющимися параллельно. Причем, надо, чтобы этот обмен не зависел от того, на каком языке написано то или иное приложение. А при работе в сети компьютеров, использующих разные платформы (Windows, Unix, Solaris и др.), желательно обеспечить и независимость общения от платформы.

Простейшими средствами общения, появившимися еще на заре Windows, являются разделяемые файлы (файлы, к которым одновременно могут получать доступ разные приложения), буфер обмена Clipboard, доступный практически всем приложениям Windows, и технология DDE — динамического обмена данными. Использование разделяемых файлов, баз данных и буфера обмена достаточно актуальны и сейчас как простейшие средства связи приложений друг с другом. А технология DDE, пожалуй, уступила место более современным способам общения.

Позднее появилась технология связывания и внедрения объектов (Object Linking and Embedding) — OLE 1. Она позволяла создавать составные документы, с которыми вы имеете дело, включая, например, в документ Word таблицу Excel. На смену этой технологии пришла OLE 2, позволявшая разным программам предоставлять друг другу свои функции. Пользуясь этой технологией, одно приложение может не просто вызвать другое, но и обратиться к отдельным его функциям, т.е. управлять им.

Следующим шагом на пути совершенствования способов взаимодействия приложений друг с другом явилась разработка COM (Component Object Model) — компонентной модели объектов. Это стандартизованное описание функций (служб) программы, к которым она дает доступ другим программам. При этом неважно, на каких языках написаны программы и где они выполняются: в одном потоке, в разных потоках, на разных компьютерах. Особенно расширяет эти возможности распределенная модификация COM — DCOM. Основой технологии COM является понятие *интерфейса*. Каждый объект COM имеет несколько интерфейсов, дающих доступ к. его функциям. Клиенту передается указатель на требуемый ему интерфейс, после чего клиентское приложение может вызывать описанные в интерфейсе функции.

На основе спецификаций COM и DCOM разработан ряд современных технологий, к которым имеют доступ приложения C++Builder и о которых подробнее сказано в следующем разделе. А завершая данный раздел, необходимо сказать еще об одном бурно развивающемся способе общения приложений — использовании глобальной сети Интернет. В Интернет могут размещаться и базы данных, и серверы, с которыми общается приложение пользователя.

Распределенные многозвенные приложения

Спецификация DCOM и созданные на ее основе технологии позволили перейти к новому этапу построения систем — распределенным многозвенным приложениям. Предшественником их явилась платформа клиент/сервер. Она сводится к тому, что на верхнем уровне имеется удаленный *сервер данных*, который хранит базу данных и осуществляет управление ею. А на нижнем уровне имеются клиентские приложения, работающие с этими данными. Это так называемые *толстые клиенты*, которые реализуют *бизнес-логику* — правила манипулирования с данными, проверки их непротиворечивости и достоверности. Впрочем, в системах клиент/сервер имеется возможность частично перенести бизнес-логику на сервер с помощью *хранимых на сервере процедур*. Клиенты могут вызывать эти процедуры со своих компьютеров и просматривать полученные ответы.

Технология клиент/сервер удовлетворительно обслуживает системы уровня отдельных подразделений некоторого предприятия. Но при создании более крупных систем уровня предприятия организация клиент/сервер сталкивается с рядом сложностей. Размещение бизнес-логики в основном на компьютерах клиентов мешает созданию единой системы с едиными правилами обработки и представления данных. Много сил

уходит на согласование работы различных клиентских приложений, на построение мостов между ними, на устранение дублирования одних и тех же функций различными приложениями.

Выходом из такого положения явилась концепция *многозвенных распределенных приложений*. Чаще всего используется трехзвенная архитектура. На верхнем уровне расположен удаленный сервер баз данных. Он обеспечивает хранение и управление данными. На среднем уровне (middlware) располагается *сервер приложений*. Он обеспечивает соединение клиентов с сервером БД и реализует бизнес-логику. На нижнем уровне находятся клиентские приложения. В ряде случаев это могут быть *тонкие клиенты*, обеспечивающие только пользовательский интерфейс, поскольку вся бизнес-логика может располагаться на сервере приложений.

Многозвенная система функционирует следующим образом. Пользователь запускает клиентское приложение. Оно соединяется с доступным ему сервером приложений. Затем клиент запрашивает какие-то данные. Этот запрос упаковывается в пакет установленного формата и передается серверу приложений. Там пакет расшифровывается и передается серверу БД, который возвращает затребованные данные. Сервер приложений обрабатывает эти данные согласно заложенной в него бизнес-логике, упаковывает и передает этот пакет клиенту. Клиент распаковывает данные и показывает их пользователю.

Если пользователь изменил данные, то цепочка их передачи на сервер БД выглядит следующим образом. Клиент упаковывает измененные данные и отправляет пакет на сервер приложений. Тот распаковывает их и отправляет на сервер БД. Если все исправления могут быть без осложнений занесены в БД, то на этом все завершается. Но могут возникнуть осложнения: сделанные изменения могут противоречить бизнес-правилам или изменения одних и тех же данных разными пользователями могут противоречить друг другу. Тогда те записи, которые не могут быть занесены в БД, возвращаются клиенту. Пользователь может исправить их или отказаться от сделанных изменений.

Для обмена информацией сервера приложений с клиентами и серверами БД разработаны различные протоколы и средства: DCOM, CORBA, MIDAS, MTS, OLEnterprise, J2EE, TCP/IP, HTTP, XML.

О COM и DCOM уже говорилось. CORBA является одним из самых мощных инструментов создания распределенных приложений. Для использования CORBA на каждом компьютере, содержащем приложения клиентов, должен быть установлен брокер VisiBroker, обеспечивающий связь приложения со средой CORBA. А на одном из компьютеров в сети должны быть установлены элементы этой среды. Если все это сделано, открываются очень широкие возможности для создания сложных распределенных информационных систем.

Система MIDAS позволяет создавать распределенные приложения, использующие самые разные протоколы и технологии: DCOM, CORBA, OLEnterprise, HTTP, сокеты TCP/IP, MTS и другие.

Система Microsoft Transaction Server (MTS) — это еще одна система создания многозвенных приложений. Правда, она работает в полную силу далеко не во всех версиях Windows.

О TCP/IP и HTTP, применяемых в Интернет и корпоративных сетях, основанных на тех же протоколах, что Интернет, будет немало сказано далее. Там же вы найдете сведения о языке HTML, используемом для описания документов Web.

Язык программирования Java — это объектно-ориентированный язык, используемый, прежде всего, для приложений Интернет. Он характеризуется простотой, надежностью, способностью создавать простыми средствами интерактивные сетевые программы. Отличительной особенностью Java является возможность исполнять созданный код на любой из поддерживаемых платформ. Достигается это тем, что

программы транслируются в некоторое промежуточное представление, называемое *байт-кодом*. Этот байт-код может затем интерпретироваться в любой системе, в которой есть среда выполнения Java. Обычно программы на языках, использующих интерпретацию, работают медленно. Но к Java это не относится. Байт-код легко переводится непосредственно в машинные коды, что позволяет достичь очень высокой производительности.

Дальнейшим развитием возможностей, заложенных в Java, явилась платформа J2EE (Java 2 Enterprise Edition). Она вобрала в себя все технологии, наработанные к этому времени, и стала основой для большинства серверов приложений. Эта платформа обеспечивает выполнение приложений, написанных в рамках определенных стандартов, и снимает с разработчика заботу об управлении очередями, транзакциями, сообщениями и многом другом.

Перечисленные технологии могут использоваться в различных видах распределенных приложений, работающих с базами данных. Но, конечно, в настоящий момент основной упор при разработке распределенных приложений делается на использование Интернет, корпоративных сетей, основанных на тех же протоколах, что и Интернет, а также технологии Web. Глобальность Интернет делает эту сеть незаменимой при создании многопользовательских распределенных приложений.

Переносимость данных и программ

Одна из важнейших проблем, решаемых в настоящее время, — переносимость программ и данных между платформами. Переносимость приложений между разными аппаратными платформами на уровне исходных кодов давно решена во многих алгоритмических языках, и, прежде всего — в C. В настоящее время основанная на C операционная система Unix реализована практически на всех аппаратных платформах. При этом исходные коды программ, написанных на C, без особых проблем переносятся на разные платформы и только иногда желательна некоторая настройка при перекомпиляции.

Но сейчас требуется большее — переносимость на уровне исполняемых кодов. Т.е. надо, чтобы одна и та же программа без дополнительной перекомпиляции могла выполняться под управлением Windows, Sun Solaris, IBM AIX и т.п. Эта задача решается средствами Java — языка программирования, коротко описанного в предыдущем разделе. Реализация байт-кода и виртуальных машин для его выполнения на современных аппаратных платформах обеспечивает для многих приложений достаточную эффективность выполнения.

Наряду с потребностями переносимости программ имеется, даже, пожалуй, более насущная потребность переносимости данных. Ведь до сих пор во многих случаях приходится поддерживать и по возможности модернизировать старые приложения DOS только потому, что написанные с их помощью документы невозможно прочитать иными способами.

Решающим шагом на пути решения этой проблемы стал в свое время язык HTML. Он по праву завоевал весь мир и стал основой построения документов Web. Но со временем проявилась недостаточность возможностей этого языка. Он стал развиваться, в него вносилось множество дополнений, а в итоге он потерял свою стройность, целостность и главное — переносимость. Дальнейшее развитие направления, начатого в HTML, вылилось в создание языка XML (Extensible Markup Language) — расширяемого языка разметки гипертекстов. Гипертекст —это то, с чем все знакомы по справкам Windows, в которых, щелкая на ссылках в тексте, вы вызываете ту или иную тему. Те, кто использует Интернет и WWW, знакомы с аналогичной особенностью любых страниц Web. Язык XML, наряду с HTML, может использоваться для описания подобных гипертекстовых документов. Но в действительности этот язык — нечто большее. Это средство разработки пользователем своих собственных языков описания гипертекстовых документов. Созданный с помощью XML язык разметки может отражать специфические

потребности конкретной фирмы или пользователя. После своего описания, такой специализированный язык может использоваться, наряду с HTML, для описания самых различных документов.

XML и его наследники обеспечивают в настоящее время не только возможности создания переносимых документов, но и универсальный способ обмена сообщениями между приложениями. На основе XML были разработаны такие широко используемые протоколы, как SOAP, UDDI, WSDL, ebXML и ряд других.

Язык Java оказался прекрасным средством работы с документами XML. Так что в настоящее время сочетание Java и XML является основой создания переносимых приложений и данных.

Сетевые службы

В настоящее время идет интенсивная работа над новым направлением, названным Web Services — сетевые службы. Не надо путать Web Services с имеющимися в настоящее время службами Web. Web Services — это новый комплексный подход к взаимодействию приложений. Он основан на протоколах, стандартах и языках, разработанных на основе XML. Верхний уровень обслуживается языком WSFL — Web Services Flow Language. На этом языке составляется описание каждой новой службы Web. Созданная служба фиксируется в некоем реестре, построенном по протоколу UDDI — Universal Description, Discovery & Integration. Этот протокол позволяет авторам публиковать и редактировать свой службы.

Следующий уровень — WSDL (Web Services Description Language). На этом языке разработчик новой службы описывает посылаемые и получаемые сообщения. Эти описания содержат спецификации набора операций, необходимых для вызова службы, и передаваемых параметров.

Пользователь, в соответствии с описанием данной службы на WSDL, может создавать адресованные службе сообщения, построенные на основе стандарта SOAP — Simple Object Access Protocol (простой протокол доступа к объектам). Получив сообщение на SOAP, служба расшифровывает его, выполняет и посылает пользователю результаты в том же стандарте SOAP.

SOAP, в отличие от большинства других аналогичных средств, предоставляет полностью открытый способ общения в Интернет, не зависимый от платформы. В качестве способа представления информации используется XML. Открытость SOAP обеспечила возможность разработки на его основе ряда технологий, в частности, упомянутого выше UDDI и Disco (Discovery protocol — протокол обнаружения), используемого для поиска служб на определенном сервере.

Распределенные приложения, построенные с использованием современных технологий Web Services, обеспечивают интеграцию приложений различных участников единого бизнес-процесса. Подобная интеграция называется приложениями business-to-business или B2B. Например, имеется компания, производящая некоторый продукт, компании, являющиеся ее поставщиками и компании-клиенты. Все эти компании имеют свои информационные системы, свои базы данных, свои бизнес-правила. Для организации эффективной совместной работы необходимо интегрировать приложения различных компаний в единую систему. Именно такую интеграцию призваны осуществлять сетевые службы Web Services.

Одной из реализаций сетевых служб является разрабатываемая фирмой Microsoft платформа .NET. Реализуется .NET на новом языке C# (Си-диез). Он похож на Java, но содержит механизмы, специфические для систем объектно-ориентированного программирования, таких, как Delphi, C++Builder, Visual J++ Microsoft. Вершиной платформы .NET являются службы Web. Это постоянно пополняемый набор служб, предназначенных для электронной коммерции и для приложений класса business-to-business. Ниже расположен уровень оболочек и библиотек, наиболее интересный для

пользователей. В него входят ASP.NET — активные серверные страницы, ADO.NET — модификация технологии ADO для баз данных, Windows Forms для работы с графикой. В качестве стандартов обмена используются SOAP и WSDL.

Средой разработки, позволяющей создавать и отлаживать программы, является Visual Studio.Net. Она поддерживает языки Visual C++, Visual Basic, C# и позволяет сторонним фирмам подключать свои инструментальные средства и компиляторы различных языков.

Платформа .NET использует новую компонентную модель, отличающуюся от используемых в настоящее время стандартов: собственной разработки COM, CORBA — разработки Object Management Group и J2EE — разработки Sun. С помощью .NET создается сборочный модуль, содержащий классы и описание интерфейса. В этом одно из существенных отличий от моделей COM и CORBA, требующих описание интерфейсов на специальном языке IDL. Сборочный модуль .NET содержит в себе всю необходимую информацию и не требует дополнительных описаний. Основным достоинством новой модели по сравнению с прежними — ее простота. Впрочем, имеется возможность создавать сборочный модуль и на основе существующей модели COM.

Есть надежда, что развитие служб Web позволит ликвидировать разрыв, существующий в настоящее время между разработкой программ для Windows и созданием сайтов Web. Страницы Web станут программами, и в то же время программа для Windows легко сможет стать страницей Web. Существенно и то, что страницы Web становятся не просто интерфейсом между программой и пользователем, но и интерфейсом API, т.е. интерфейсом между программными модулями.

C++Builder 6 и его место в семействе программных продуктов Borland

После того, как вы познакомились в предыдущих разделах с некоторыми основами современных информационных технологий (или если вы все это знали ранее), можно начать говорить об основном предмете — C++ Builder 6. Это мощная система визуального объектно-ориентированного программирования, позволяющая решать множество задач, в частности:

- Создавать законченные приложения для Windows самой различной направленности, от чисто вычислительных и логических, до графических и мультимедиа.
- Быстро создавать (даже начинающим программистам) профессионально выглядящий оконный интерфейс для любых приложений, написанных на любом языке. В частности, с помощью C++ Builder можно объединить в единую систему с удобным современным интерфейсом имеющиеся на предприятии при кладные программы DOS, Windows и др. Интерфейс удовлетворяет всем требованиям Windows и автоматически настраивается на ту систему, которая установлена на компьютере пользователя, поскольку использует многие функции, процедуры, библиотеки Windows.
- Создавать мощные системы работы с локальными и удаленными базами данных любых типов. При этом имеются средства автономной отладки приложений с последующим выходом в сеть.
- Создавать многозвенные распределенные приложения, основанные на различных технологиях.
- Создавать приложения, которые управляют другими приложениями, в частности, такими программами Microsoft Office, как Word, Excel и др.
- Создавать кросс-платформенные приложения, которые можно компилировать и эксплуатировать как в Windows, так и в системе Linux.
- Создавать приложения различных классов для работы в Интернет и в

интранет.

- Создавать профессиональные программы установки для приложений Windows, учитывающие всю специфику и все требования Windows.
- И многое, многое другое, включая создание отчетов, справочных систем, библиотек DLL, компонентов ActiveX и т.п.

C++ Builder — чрезвычайно быстро развивающаяся система, так как ее создатели постоянно отслеживают все тенденции информационных технологий. Новые версии выпускаются ежегодно. Только выпуск последней версии немного задержался, но это понятно, если оценить, сколько нового в нее введено.

Хотелось бы коротко остановиться на других программных продуктах фирмы Borland, чтобы можно было оценить общую стратегию развития и место, которое занимает в ней C++Builder. Прежде всего, надо отметить старшую сестру C++ Builder — Delphi. Это тоже система визуального объектно-ориентированного программирования, которая появилась несколько раньше C++Builder. Внешне Delphi является точной копией C++Builder, имеет тот же набор компонентов, те же страницы библиотеки, тот же инструментарий. Но Delphi использует не язык C++, а язык Object Pascal. Новые версии C++Builder выходят параллельно с версиями Delphi, но сдвинутые во времени примерно на полгода. При этом новые версии Delphi обычно содержат много нового, а версии C++Builder реализуют то, что введено в очередной версии Delphi, и добавляют нечто новое, что в дальнейшем будет добавлено в очередной версии Delphi. Например, версия C++Builder 5 мощнее версии Delphi 5, но значительно слабее, чем Delphi 6. А версия C++Builder 6 несколько более мощная, чем Delphi 6.

Для большинства применений возможности эквивалентных версий C++ Builder и Delphi примерно одинаковы. Это не удивительно, поскольку язык Object Pascal в настоящее время очень близок (конечно, если не учитывать синтаксис) к C++. Так что для большинства задач выбор C++ или Object Pascal, и, соответственно, C++Builder или Delphi — дело вкуса и привычки программиста. Но все-таки язык C++ пока несколько более мощный, чем Object Pascal. К тому же в C++ наработаны более обширные библиотеки функций. Так что некоторые, сравнительно сложные задачи проще решать с помощью C++Builder.

Среди традиционных продуктов Borland надо также упомянуть JBuilder — систему визуального проектирования на языке Java. А JBuilder — система, аналогичная C++Builder, но использующая Java, а не C++.

В последнее поколение программ Borland, наряду с C++Builder 6 и Delphi 6, вошла и новая разработка фирмы — система визуального проектирования Kylix. Она предназначена для разработки приложений Интернет, настольных приложений и приложений баз данных в операционной системе Linux. Эта ОС в последнее время становится основной платформой для создания приложений Интернет и начинает конкурировать с Windows в настольных операционных системах.

Внешне среда разработки Kylix выглядит так же, как среда C++Builder или Delphi. И методика работы с ней та же, что и в C++Builder. Набор компонентов аналогичен библиотеке CLX в C++Builder 6, о которой будет сказано позднее. Так что создание Kylix — подарок всем, работающим в Linux.

Если говорить о соотношении C++Builder и Kylix, то надо отметить, что C++Builder 6 тоже сейчас позволяет строить кросс-платформенные приложения, которые могут компилироваться как для Windows, так и для Linux. Для этого в C++Builder 6, наряду с традиционной библиотекой компонентов VCL, включена библиотека CLX (cross-platform component library). Эта библиотека эквивалентна той, которая имеется в Kylix. Так что приложения для ОС Linux теперь можно создавать и с помощью C++Builder.

Основное внимание в последней генерации своих программных продуктов фирма Borland уделила созданию средств проектирования распределенных информационных систем уровня управления предприятием. Здесь, прежде всего, надо отметить сервер

приложений Borland AppServer 4.5. Он использует сочетание J2EE (Java 2 Enterprise Edition) и CORBA. Таким образом, богатые функциональные возможности J2EE дополняются средствами коммуникации CORBA, обеспечивая все необходимое для формирования эффективных распределенных систем в корпоративных сетях.

Поскольку AppServer использует Java, он естественным образом интегрируется с JBuilder. Но нет никаких препятствий для создания клиентских и серверных приложений на C++Builder, которые предназначены для последующей работы с AppServer.

Таким образом, C++Builder 6 органично вписывается в семейство программных продуктов Borland, отслеживающее, а во многом и формирующее новейшие тенденции информационных технологий. И из прекрасного средства создания приложений для Windows C++Builder превращается в инструмент создания приложений для многозвенных распределенных кросс-платформенных корпоративных информационных систем.