Delphi Client/Server Suite 2.0
Open Architecture

Richard Morris
KHIRON *Software* PTY LTD
Borland International
July 2, 2025

Delphi Overview

Since its introduction in February 1995, Borland's Delphi Client/Server has set a new standard in high-performance rapid application development. As a result of Delphi's unique combination of a native code compiler, visual two-way tools and scaleable database technology, Delphi has achieved numerous awards worldwide and has made best seller lists around the world. Delphi has also achieved a tremendous level of third party support including dozens of add-on libraries and compatible tools, over 30 books, a half dozen monthly magazines and newsletters, and many training courses supported by a growing number of third party consultants.

Borland has a forthcoming second generation 32 bit Delphi environment, known as Delphi Client/Server Suite 2.0, for Windows NT and Windows 95. This 2.0 release incorporates several new technologies in order to further improve the productivity of developers and the performance of applications. The Delphi Client/Server Suite 2.0 is based on a new 32 bit optimizing native code compiler offering three to four times greater performance than before, and a new 32 bit Borland Database Engine with a faster, scaleable query engine. An expanded Open Tools API allows Delphi Client/Server Suite 2.0 to interface with third party tools and libraries to customize the development environment thereby increasing overall productivity.

The Case for Integration

There is no doubt that Delphi has achieved worldwide acceptance as a leading edge development tool. It has a unique combination of a powerful native code compiler, a scaleable and extensible database architecture, and a flexible component development paradigm that increases productivity and increases return on investment. However to paraphrase an old saying '*No tool is an Island*'. Corporate customers need to be able to integrate with a variety of tools and technologies.

Delphi includes a rich suite of such tools such as an Editor, Debugger, Visual Form builder, Compiler, Linker, Resource Builder, and Project Manager. For many projects this suite provides everything you would need to jump right in and start developing great code.

However, more complex client/server projects in large organizations typically have differing requirements for specialized tools. The number and variety of such tools is almost as varied as the developers and teams who use them. They can range from custom built suites that model the organizations procedural model, to off-the-shelf solutions from Developer tool vendors. These fulfill purposes such as CASE (Computer aided systems engineering), Object Oriented Analysis and Design, Version Control, Code analyzers and various software metric tools, testing and quality assurance tools, and more.

In this paper we are going to review how Delphi integrates with the latest leading edge technologies, and 'tried-and-true' software development tools to provide developers with an environment unparalleled in PC software development. Delphi itself was written in Delphi, and Borland has exposed interfaces into the environment's internal structures, called the Open Tools API, which allow organizations to provide tight integration with their preferred suite of tools. Finally we will see how some of the industry's most prominent developer tool vendors have integrated their respective technologies with Delphi. At the end of this paper is an appendix that briefly documents the published interfaces into Delphi to give you a head start in developing your own integration solutions.

I Delphi OpenTools API

Delphi Client / Server Suite 2.0 includes an OpenTools API that allows developers to customize their environments with third party technologies and thereby increase their productivity. The API includes:

- *Design Interface for creating custom object property editors*
- *Tools Interface for generic tool integration*
- *Expert Interface for creation of easy to use experts*
- *Version Control Interface for Source Control Management in a team environment*
- *File Interface for custom file management*

Delphi provides a well defined Open Tools API that allows developer of custom technologies to integrate into the Delphi development environment. Delphi provides a series of interfaces so that any developer can build their own tools in Delphi and integrate them seamlessly into the environment.

Today, vendors of custom tools for developers are capitalizing on the Open Tools API by integrating their solutions directly into the Delphi environment. The OpenTools interface comprises: a Design Interface, a Tools Interface, an Expert Interface, a Version Control interface, and new in Delphi Client/Server Suite 2.0, a File Interface. We shall cover them in further detail later and the Appendix covers the technical interface more clearly.

- Delphi 2.0 CLIENT/SERVER Suite's **Design Interface**: The Delphi environment pre-defines property editors for the standard components, however productivity improvements are available by allowing developers of custom components to create custom property editors using the Design Interface.

- The **Tools Interface** provides the developer an object class to be used for generic tool integration. This class contains interfaces for project and file management, dialogs, the Visual Component Library (VCL), and exception handling for external tools.

- The **Expert Interface** allows developers to easily create their own Experts (also known as "Wizards") which simplify or automate tasks. An expert could do almost any task, such as importing proprietary data into the Scaleable Data Dictionary, automatically creating forms, automatically creating application frameworks, or generating reusable forms that are shared in the Object Repository.

- The **Version Control Interface** has functions that facilitate source code management in a team environment. The Version Control Interface has been used to integrate industry standard tools such as Intersolv PVCS, now included in Delphi Client/Server Suite 2.0, and MKS Source Integrity. Both of these tools make use of custom menus and project hooks in the Version Control Interface to provide easy access to the underlying version control software's functionality which leads to greater team development practices.

- Delphi Client/Server Suite 2.0 includes an expanded **File Interface** allowing developers complete control in how they integrate with all aspects of file management. For example, third parties can replace the traditional PC file system, and extract Delphi project code from Databases, archive files, or third party applications.

II Components & Classes

Delphi Client / Server Suite 2.0 incorporates an object oriented component model that allows for maximum code reusability and maintenance.

- *Developers create components from within the Delphi Environment*

- *A large number of Third Party custom components are readily available to provide specific functionalities*
- *Specialized technology can be integrated into the Delphi Client/Server Suite 2.0 environment*

Delphi empowers developers to achieve significant productivity through its Visual Development environment. The Visual Development Environment allows a developer to simply grab a pre-built component and drop it into an application. There are over 100 standard components that ship with Delphi Client/Server Suite 2.0. Additionally Delphi can turn any OLE Control (OCX)into a custom component providing open access to a growing third party market. Because Delphi is an object-oriented development environment, it is easy to create your own custom controls or subclass OLE controls via inheritance thereby extending the power and functionality of the environment to meet the changing business needs.

The Delphi environment manages these components through the Visual Component Library (VCL). Developers can group their components into categories, and easily find them through the intuitive tab interface.

Figure 1- Delphi Client/Server Suite 2.0 has a tabbed interface to an extensible palette of over 100 components

A key feature of Delphi is the ability to create new components as necessary from within the Delphi environment instead of having to create components within another tool altogether. These new components, through object-oriented inheritance, can be as simple as an existing component with some added functionality, or a completely custom component that is based entirely on new code. Additionally, Delphi has an expert to build the skeleton structure of a component with a single click. Customized components meet the changing needs of the development team and can be reused in many different applications thereby increasing productivity and maintainability.

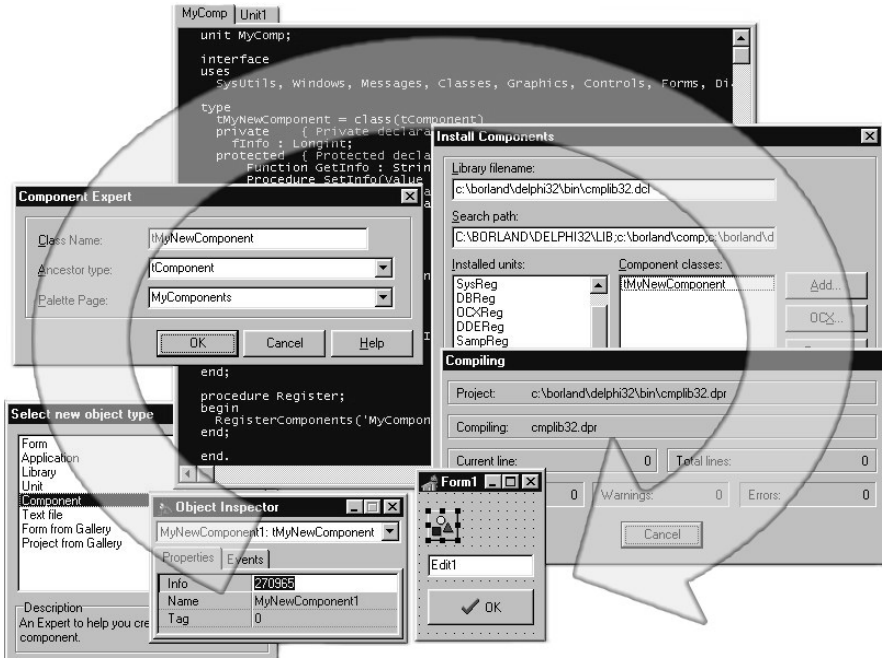

Figure 2 - Delphi Component Design in action

The significance of being able to create new components in Delphi is three fold:

- Developers don't have to switch to another development environment when creating new components.
- Delphi teams build reusable components libraries so that they can create more applications resulting in exponential productivity growth.
- Components based application can be easily maintained.

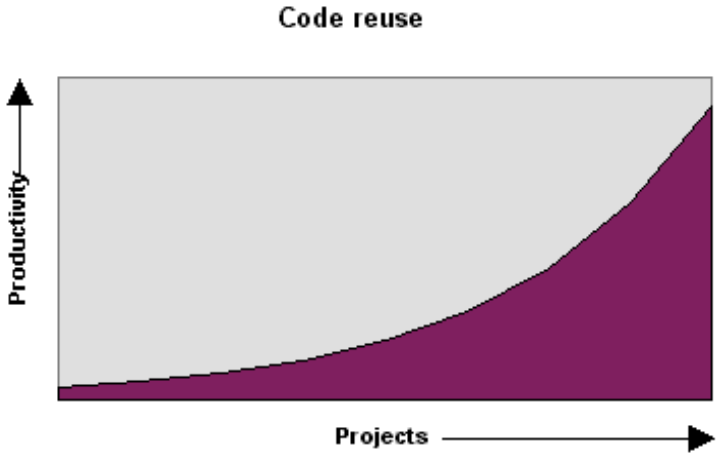

Figure 3 - Code reuse promotes exponential productivity growth over time

A Third Party Custom Components

As a developer you want to enhance your productivity leveraging the expertise of specialists, the best method to do this is to utilize custom component written by experts. The uses of a custom component are as varied as the uses of a computer: from telephony to virtual reality. There are many established third party companies building Delphi components to provide developers with specific functionalities such as Document Image Control or access to specific external hardware.

The following table shows a few of the many Delphi specific components available from component specialist providers . Of course there are more than 600 OLE Control (OCX) vendors that make components that can also be incorporated directly into Delphi. Finally there are hundreds of components available on both the Internet and CompuServe.

Product	***Description***
ABC for Delphi	Professional exception handling and error logging components.
Accounting for Delphi	Accounting Components
Apiary Developers suite for Netware	Native Delphi encapsulation of Netware API's to take advantage of Netware 4.1 features such as Auditing, Netware Directory Services, Extended Attributes, Fileserver Environment, File System, Messaging, Printing, Queue management etc.
App Enhancement Component Pack	Series of Delphi components for enhancing screen design.
Async Professional for Delphi	Asynchronous communications components
DrawKit	Delphi graphic components
DynaZip	Data compression toolkit.
ImageBASIC	Document Imaging components (OCR, Scanners, TWAIN)

ISG Components	Components for Email and Forms enabling applications.
King Calendar Professional Edition	Calendar components for creating scheduling and PIM applications.
Light Lib Business DrawKit	Object Oriented Delphi controls for charting such as business Graphs with drill down capability.
Light Lib Images	Object Oriented Delphi controls for Image manipulation including full scanner (TWAIN) support.
ReportPrinter	Native Delphi Report components
Orpheus	Components for designing validating Data entry applications with comprehensive collection of productivity components for Delphi
ZMaster	Multi-lingual drop in components for English and French language applications

Table 1 - Sample selection of Custom components[1]

Almost any enabling technology can be wrapped into Delphi components, and healthy growth of the add-on component market is testimony to: a) demands by the market place for component based solutions and b) demands for integrated solutions with Delphi.

B Custom Component and Property Editors

Another important feature in Delphi is its ability to customize not only the components that a developer builds, but the context and mechanisms with which developer edits those components. Custom component are obviously a first step toward the integration of specialized technology into Delphi projects, however in a lot of instances it is in the editing and binding of the properties of Custom controls that significant customization can be achieved. For example in developing with Delphi Client/Server Suite 2.0 it is a common requirement to build complex SQL (Structured Query Language) definitions to access data from Remote servers. Borland has included technology called the Visual Query Builder (VQB) from Integra to allow a developer to visually customize the SQL statement property of a Query component (Figure 4).

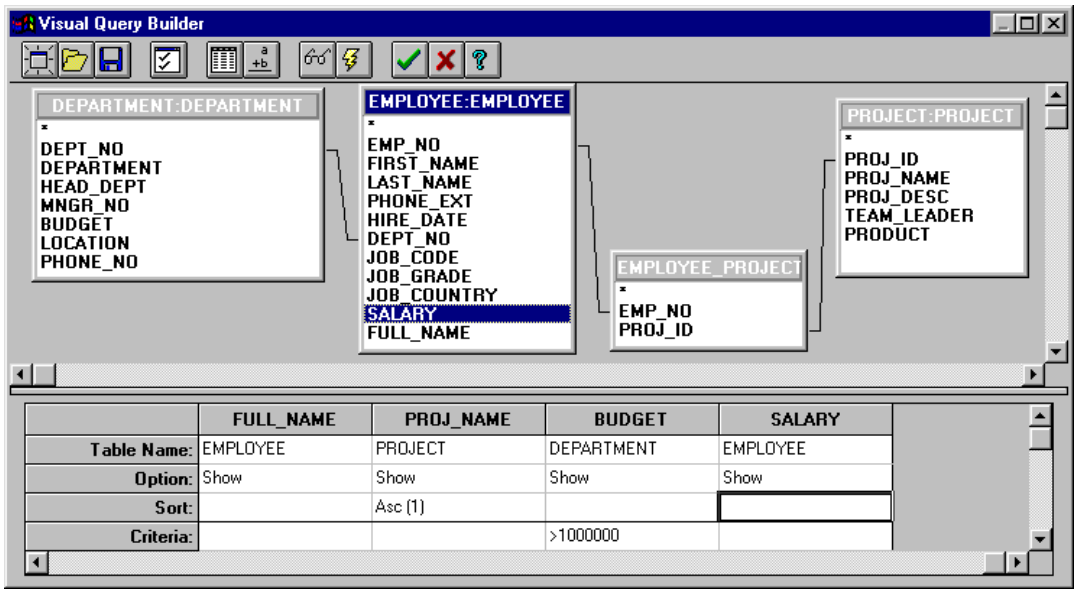

Figure 4 - The Visual Query Builder is completely integrated into Delphi Client/Server Suite 2.0

[1] This is just a sample of some custom components, as there are many emerging on the market every day. You may wish to contact your local Borland Office for the latest edition of the *PowerTools* catalogue which is a more comprehensive list including contact details for these companies.

By editing the SQL property of a Delphi Query Component you have access to a powerful, custom tool for building complex queries. Visually build complex data relationships through easy to use *'point and click'* technology. The SQL code (Figure 5) is automatically put into the SQL property of our Query component. Without this custom component editor you would have to type that SQL code into the Components SQL property by hand each time.

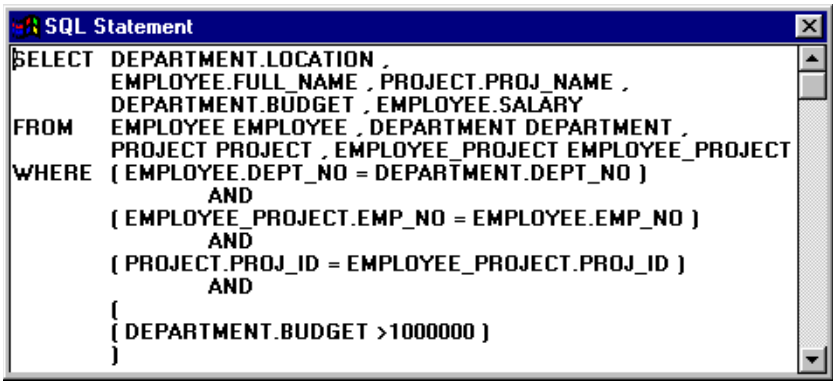

Figure 5 - SQL Code produced by the Visual Query Builder

Delphi provides developers a lot of control over how it handles custom components by allowing developers to build their own experts to speed common development exercises and procedures. The Design Interface which is part of the Open Tools interface of Delphi allows developers to handle how any property is edited within the Delphi environment.

III Data Connectivity and Custom Data access technology

Delphi Client / Server Suite 2.0 allows access to data through high speed SQL Link Native Drivers, ODBC, and through customized solutions. Data access is available for

- *Native access to: Oracle, Sybase, Informix, MS SQL Server, InterBase*
- *ODBC access to (for example): AS400, IMS, DB2, Lotus Notes...*
- *Custom access to: BTrieve, Access, OLAP, OODBMS, TPM...*

By using industry standards, such as SQL92 and ODBC, Delphi is able to access a broad range of data sources. However for pure performance the Borland Database Engine incorporates high speed native SQL Links drivers to commonly used RDBMs such as Sybase, Oracle, Informix, Microsoft SQL Server, and InterBase.

Since the release of Delphi, a large number of data specialist companies have developed custom technologies enabling Delphi to access a broad range of data formats, from popular PC formats such as BTrieve to RDBMS's, OODBMS's, OLAP's, etc. There are also technologies that support Transaction Processing Architectures enabling multi-tier applications and other technologies enabling Delphi to integrate legacy mainframe applications with new development projects.

Company	*Product*	*Description*
Gerald Limited	AS400RemoteCommand AS400DirectTransfer	Native Delphi VCL Components for high performance native access to AS400 data
Brainstorm Technologies	Delphi-Link for Notes	Native controls to enable Delphi Developers to build mission critical applications using Lotus notes
AmiSys Inc.	Titan	Database Engine allowing developers to access BTrieve data files natively

Successware	Apollo	Database engine for Foxpro, Visual dBase, and HiPerSix indexed data files
Innovative Solutions and Technologies	OpenExchange	Data import-conversion-export tools
AT&T GIS	Top End	TOP END is a family of open software products that offers robust client/server middleware services for the distributed enterprise environment.
Digital Equipment Corporation	ObjectBroker	Digital's compliant implementation of the Object Management Group's (OMG) Common Object Request Broker specification.
Open Horizon	Connection Application Broker	Provides Delphi with the ability to support 3-tier architectures (Transarc's Encina, BEA System's Tuxedo) transparently and allowing Delphi applications to be deployed into a stable 3-tier architecture.
Attachmate	QuickApp	Delphi samples and technology to access existing applications running on mainframes, and integrate them into new projects
Sequelink	Sequelink 4.5	Enables Delphi through the BDE to access data from RDB, DB2, AS400, Ingres, Teradata, Adabase and other database systems.

Table 2 - Sample selection of Custom Data access technologies[2]

The OLE extensions in Delphi Client/Server Suite 2.0 provide the developer with an even greater selection of data sources including: Spreadsheets, Business Objects encapsulating complex business rules, and Direct Access Objects. Delphi was built to simplify the process of developing flexible applications that work with complex data requirements.

IV Delphi Experts using the Open Tools API

Delphi Client / Server Suite 2.0's Open Tools API allows developers to create experts that automate work flow. Numerous third party vendors are also providing experts that automate the Delphi environment. Delphi Client / Server Suite 2.0 also includes experts that:

- *Create Database forms thereby increasing productivity*
- *Imports Case Data Repositories into the Delphi Scaleable Data Dictionary so that business and data models can be easily used to create application solutions*
- *Can be shared by the team via the Object Repository.*

Delphi ships with several experts (also called wizards) which automate common development tasks, such as building components, and populating data access and display forms.

The Expert Interface allows Delphi to receive from an expert module an exported Class that has the ability to integrate, remove and describe itself from Delphi. Experts are generally automation tools for Delphi that create forms and applications, although an expert could do almost any task. Experts can be invoked upon creation of a form in the Form Gallery, creation of an application in the Application Gallery, or in an ad-hoc manner from the Help menu.

[2] This is just a sample of available custom data solutions, for more current details please feel free to contact your local Borland Office for the latest edition of the *PowerTools* catalogue which is a more comprehensive list including contact details for these companies.

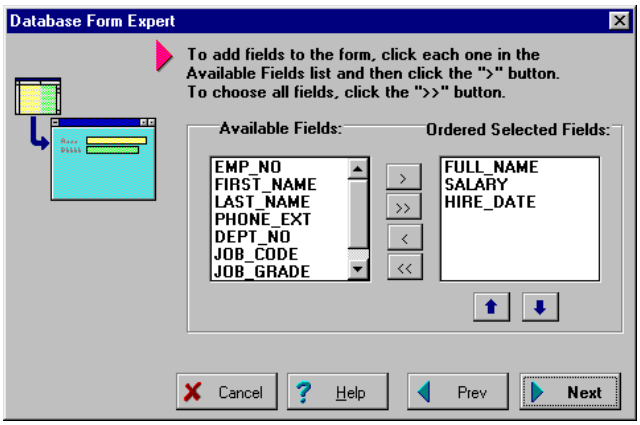

Figure 6 -Delphi Database form Expert in action

Delphi Client/Server Suite 2.0 also includes an object repository for storing standard Dialogs, Forms, and experts. As you can see in (Figure 6) the database form expert appears as an available form in our Object repository (Figure 7). From this point a developer can copy, use or inherit a previously designed form class such as an about box, or select an expert to generate a brand new form class.

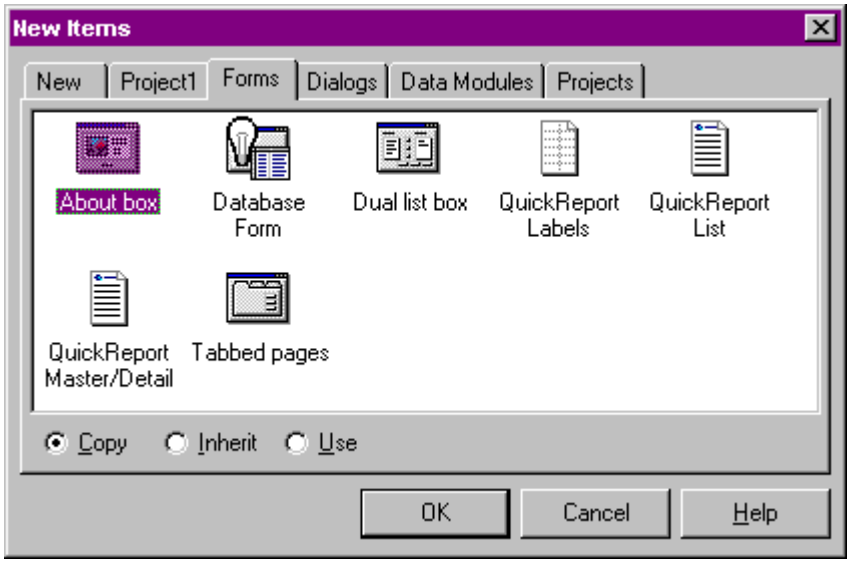

Figure 7 - The Delphi new form Repository

A Custom Expert Technology and Productivity tools

There are a number of useful Expert based solutions commercially available for Delphi including a number of installation experts that automate the production of deployment builds and Install disks for Delphi applications. Such products include InstallShield from Stirling Technologies, Eschalon Setup Pro from Eschalon Development Inc., and WISE Install from Great Lakes Software.

Product	***Description***
Conversion Assistant	Automate Conversion of Visual Basic projects to Delphi
Component Create	Tools for designing and creating new customized native components.
Eschalon Setup Pro	Create Installation builds including custom BDE installs

Help Magician Pro	Visual hypertext help builder
Multi Edit	A drop-in customizable replacement for the standard Delphi code editor
The Generator	Data Testing tool for generating Test data.
PC-Install	Create Installation builds
PolyG111	Multi-lingual expert for creating multi-lingual applications.
Premia Codewright	Drop in Code editor to extend Delphi
ReportPrinter	Native Delphi Report components
SQA Test	Quality Assurance testing tools for Delphi

Table 3 - Sample selection of Delphi productivity tools[3]

The technical appendix to this paper summarizes the documented interfaces used to build such Expert tools.

B The CASE Expert

Borland's Research & Development teams are continually developing custom expert solutions for upcoming releases of Delphi. We have seen examples of these in the Dialogs, and Database experts shipping with Delphi, and Resource Expert (available in the Delphi RAD pack) which allows Developers to import complex Dialogs from other Windows Development environments such as C++. In this section we shall look at the CASE expert built into Delphi Client/Server Suite 2.0.

Historically the best practices of complex system design have been automated in CASE tools, allowing systems developers to design classes and interactions, to be coded in their implementation environment. Development teams that are considering large scale systems design, or currently have an investment in CASE expertise will want to consider carefully how they can integrate their development tool of choice with the latest state-of-the-art CASE technologies.

Some CASE tools extend system documentation to automate the creation of code for specific development tools. A few development tools have technologies to directly import models designed in CASE tools into their own class repositories. The caveat is that these are traditionally static solutions for integration only between specific versions of limited tools. The Delphi R&D Team have designed a modular open solution to solve this issue, called the CASE Expert, that allows information from a CASE model to be imported directly into the Delphi Scaleable Data Dictionary.

The CASE expert uses the Open Tools API to provide an interface to allow classes to be imported from custom environments into the Delphi environment. This expert works by allowing the developer to select from a number of CASE tool specific modules (Figure 8), and then to select from projects modeled within those CASE tools, and finally to choose an integration strategy for importing those models into Delphi. This solution gives developers the flexibility to choose the CASE technology that allows them to achieve the highest productivity, and be assured of continued integration support from their CASE tool of choose.

[3] This is just a sample of Delphi productivity tools available today. You may wish to contact your local Borland Office for the latest edition of the *PowerTools* catalogue which is a more comprehensive list including contact details for these companies.

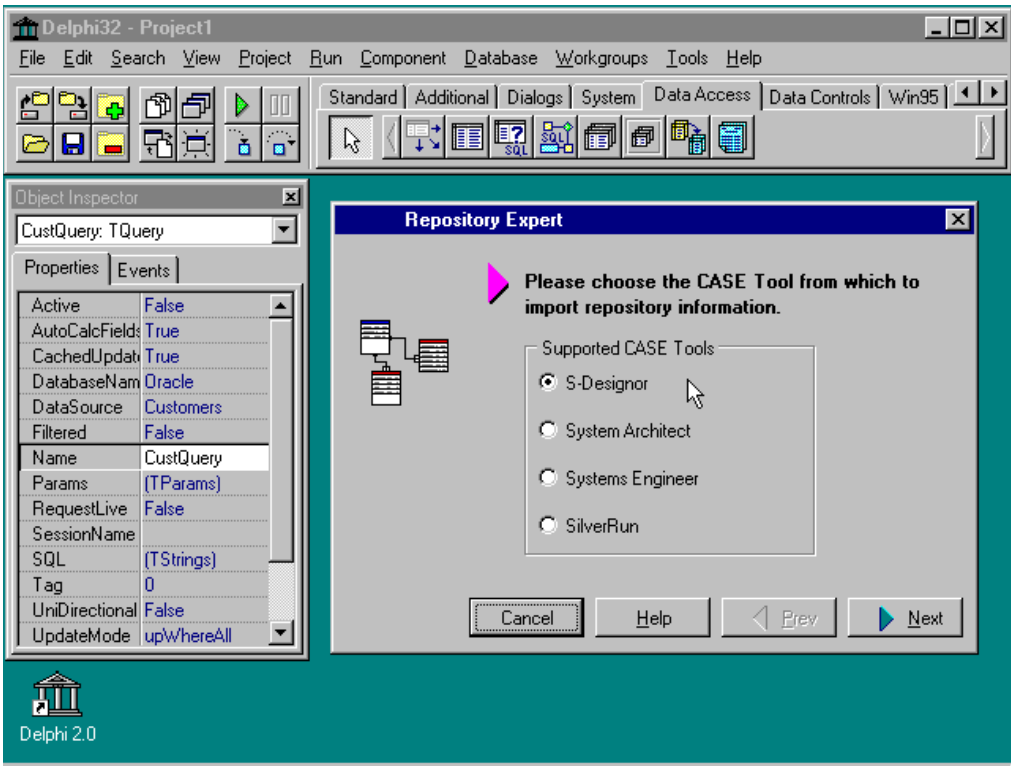

Figure 8 - CASE Methodologies integrated in Delphi Client/Server Suite 2.0 by an Expert

With the Client/Server version of Delphi 2.0, Borland has built integration modules for a number of well respected CASE tools such as LBMS Systems Engineer, Popkin System Architect, Sybase S-Designor and CSA SilverRun. Because of it's open and modular architecture further integration with other tools can be added later on.

V Microsoft systems technology

Delphi Client / Server Suite 2.0 supports Microsoft systems technology so that applications can easily integrate with desktop suites, access varying data sources and offer the highest performing applications. Delphi supports systems standards such as

- *OLE Clients, OLE Servers, Remote Automation*
- *OLE Controls, OLE Containers*
- *ODBC*
- *Threading*
- *COM*

An important advantages of Delphi is that it fully supports Microsoft standards. Delphi Client/Server Suite 2.0 is a Windows95 and NT logo application that allows developers to create logo applications passing all of the stringent requirements set down by Microsoft.

Delphi has classes for embedding Microsoft OLE objects, Microsoft DDE conversations, INI files and so on. Delphi 2.0 extends this to handle OLE Automation , both in and out-of-process servers, Windows95 component classes, the Registry, and Windows32 API for Windows NT and Windows95. Additionally Delphi representation of classes follows the Microsoft Common Object Model (COM), allowing Delphi to create and inherit from COM classes created in C++ such as Microsoft's DirectX APIs for low level hardware interaction.

Delphi Client/Server Suite 2.0 can encapsulate any OLE Control (OCX) by automatically generating a Delphi object wrapper The example (Figure 9) shows a simple Internet Browser created by dropping an HTML OCX component on a form.

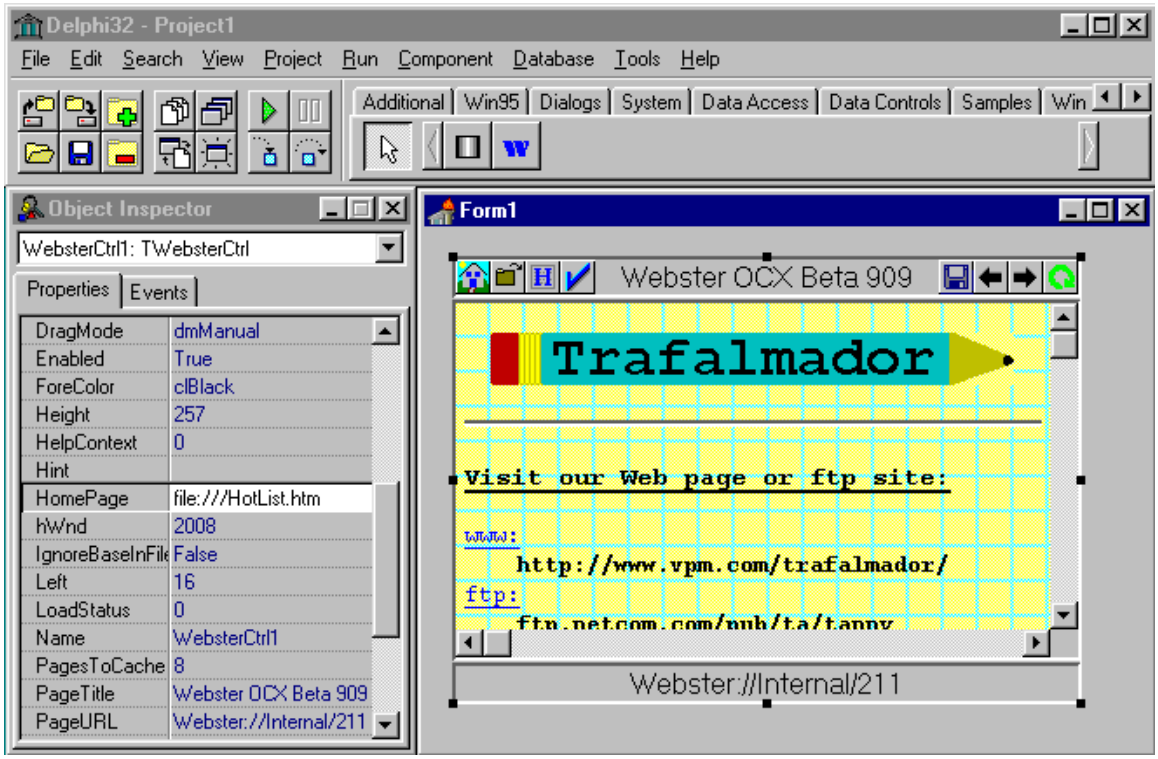

Figure 9 - Sample OCX component in Delphi 2.0

Delphi supports both the Common Object Model (COM) and OLE. Delphi allows the developer to handle both the controller and server side of OLE automation and is fully compatible with the forthcoming Network OLE technology, as well as Visual Basic 4.0's Remote Automation technology. This means Delphi will build native compiled applications on both ends of a network OLE link, allowing high performance application partitioning for mission critical applications.

By adhering closely to operating system standards, developers are guaranteed that Delphi will remain on the leading edge of development technologies, providing them a stable foundation for productive development in any office application environment.

VI Developer tools

Delphi Client / Server Suite 2.0 provides solutions from design through deployment.

- *CASE and OOAD tools increase design capabilities and work specifically with Delphi*
- *Version Control tools enable team development and reduces development chaos*
- *Transaction Process Monitors allow for high volume client server applications.*

To gain a better understanding of the Integration development occurring in the Industry today we shall look at tool integration and some specific tools that will probably be of more interest to teams in large organizations. These technologies are VCS (Version Control and Configuration management), CASE tools (Computer Aided Software Engineering), and Transaction Processing.

A Version Control Systems

Delphi Client/Server Suite 2.0 includes source code control with *Intersolv's PVCS Version Manager* to allow a team of developers to work most efficiently. Check In / Check Out capabilities supplemented with Visual Differencing, Reporting, and Archive management assists in the development of deployable applications. PVCS, the industry leading source code control and configuration management software, helps large teams of developers work together and reduce development "chaos." This in turn reduces errors and brings product to market more quickly and more profitably..

We examine the VCS interface, and two commercial VCS products that have been integrated into Delphi: PVCS from Intersolv and Source Integrity from Mortice Kern System.

Company	***Product***	***Description***
Intersolv	PVCS	Integration built by Borland and included with Delphi Client/Server Suite 2.0
Mortice Kern Systems	Source Integrity	Integration built by MKS into Source Integrity, using Open Tools API

Table 4 - Version Control solutions

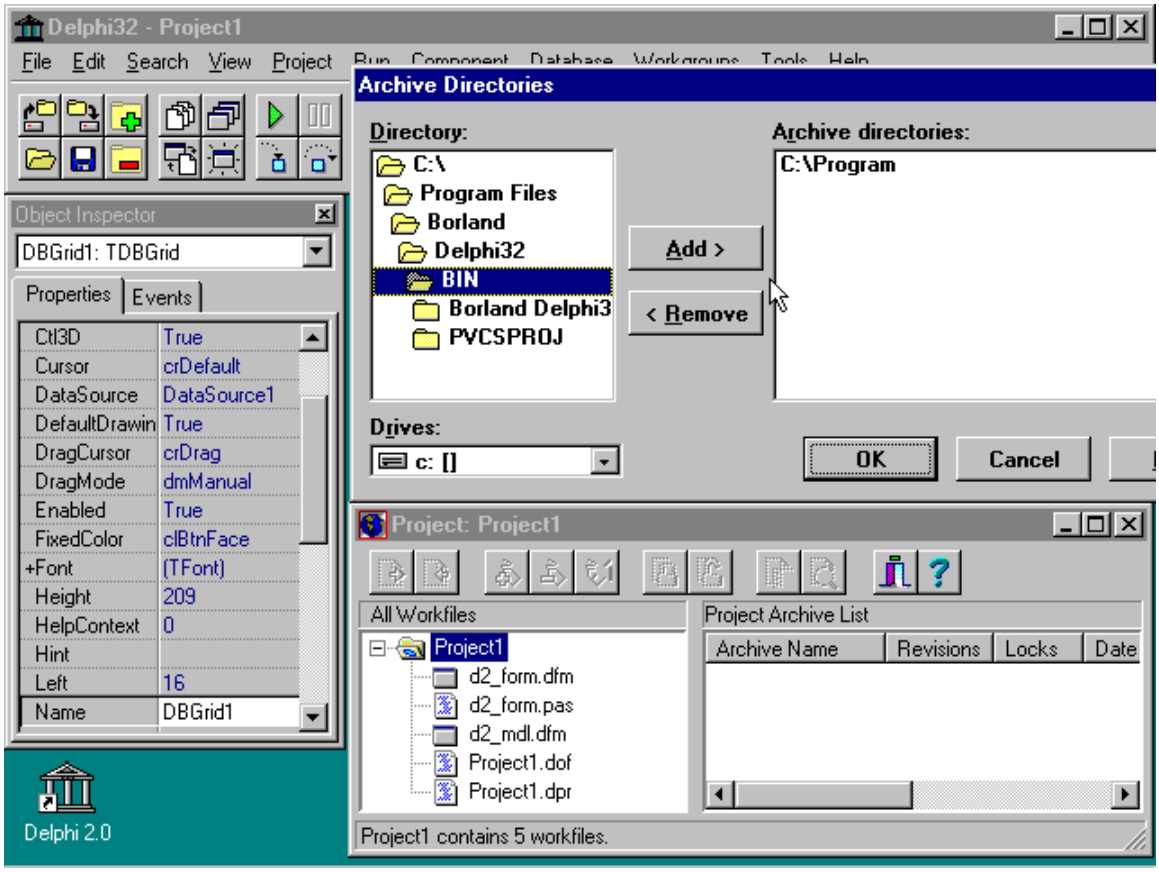

Figure 10 - Intersolv PVCS technology is included in Delphi Client/Server Suite 2.0

The Borland and Intersolv R&D teams built an interface to PVCS directly from within Delphi and documented the Interface as the VCS management portion of the OpenTools API. You can see (Figure 10) the Delphi menu Item called **Workgroups** which surfaces the ability to put and get forms and code, and the ability to mark and retrieve version information.

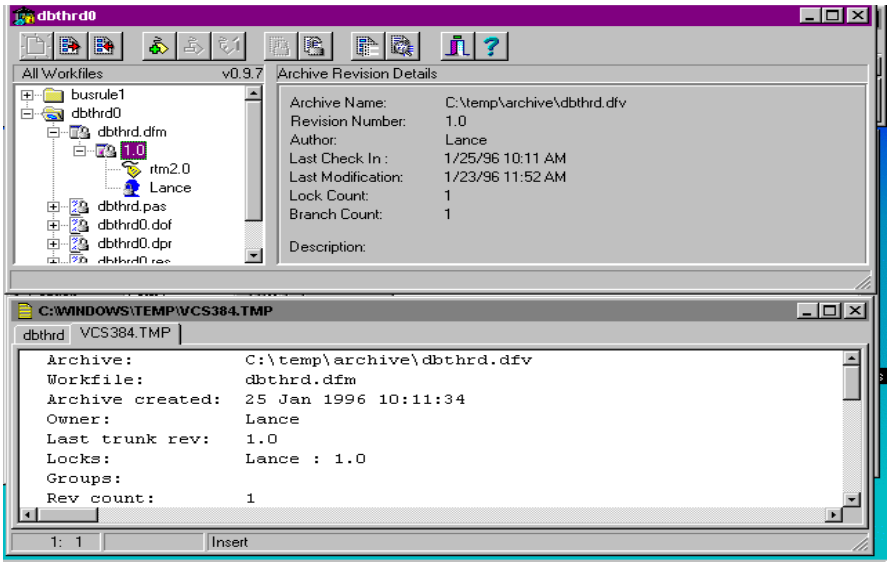

Figure 11 - Easy to use Delphi Interface to powerful Source Code Management functionality

With the release of Delphi Client/Server Suite 2.0, Borland's engineers have again raised the productivity bar to provide a hierarchical project management tool (Figure 11), tighter integration and better control of the version control repository.

The MKS Source Integrity interface was created by Mortice Kern themselves, and ships with their Product. Again you can see (Figure 12) that they have used the menu integration to add a **Source Integrity** menu item which surfaces the ability to check in and out individual files and projects from the archive, and run the Source Integrity Management tools.

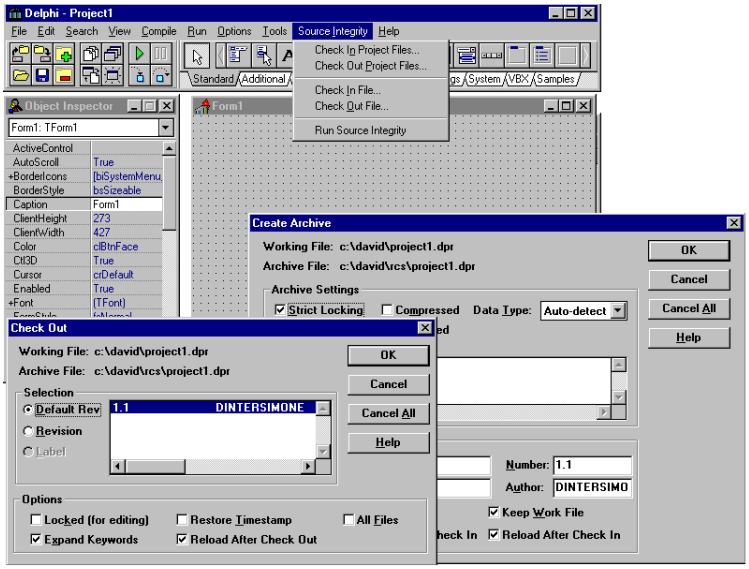

Figure 12 - Montage showing features of MKS Version control Interface

B Case tool integration

CASE Tools allow Analyst/Programmers to automate the creation of Code based on Conceptual schema. The Models used vary depending on the category of CASE tools, for example there are those that create Data Models and Business rules based on Entity relationship diagrams, State Transitions and other models of defining information flow, OO (Object Oriented) Case tools Automate the creation of Classes and interactions directly from a graphical representation of Classes, Relationships, and responsibilities from Industry Modeling standard such as the Booch93 , OMT (Rumbaugh) or Coad Object models.

We've already discussed the CASE expert in Delphi Client/Server Suite 2.0 that offers a customizable modular technology for integrating a number of CASE tools, as well there are a number of CASE vendors who have native support for the Delphi language and it's Data sources (Table 5 - CASE integration).

Company	*Product*	*Description*
Rational	Rational Rose	A forthcoming edition of this product of Grady Booch and James Rumbaugh will be integrated with Delphi.
Popkin	System Architect	Popkin are converting their product to generate InterBase schema. Borland have built a CASE expert integration module for these modules.
CSA	SilverRun	Business process application development and Database design CASE tool (Delphi and InterBase support). Integration module within the Delphi 2.0 CASE expert.
LBMS	Systems Engineer	Integration module within the Delphi 2.0 CASE expert.
SDP (Sybase)	S-Designor	Integration module within the Delphi 2.0 CASE expert.
Asymetrix	InfoModeller	Windows based data modeler, can generate data schema for Delphi.
Logic Works	ErWin	Windows based ER modeler, can generate data schema for Delphi.
Cadet	CADET	Custom built Delphi specific CASE tool built using Delphi.
Micro Gold software	WithClass	Generic CASE tool supports C++, SmallTalk and Delphi Class creation.
Other	Custom in-house CASE tools.	Delphi's Open Tools API and CASE Expert allows developers to extend Delphi to work with any custom CASE technology

Table 5 - CASE integration[4]

One of the most complex issue involved with CASE tools is the ability of the CASE tool to reverse engineer the native source back into the systems own model. There are several good data modeling CASE tools that do reverse engineer the respective databases structures that Delphi is capable of accessing (Oracle, Sybase, SQL Server, Informix, dBase, Paradox). Reverse engineering the actual Delphi classes into an OO Model is expected in the future. However for new projects, generating classes from a CASE

[4] For more current details please feel free to contact your local Borland Office for the latest edition of the *PowerTools* catalogue which is a more comprehensive list

tool that are then sub-classed to enter the application allows the CASE tool to be reused iteratively within the development cycle.

Popkin Software have announced SA/Delphi link to allow Delphi to utilize System Architects Object Repository technology to store data elements, entities/tables and entire Delphi forms. This is a precursor to an upcoming version that will be able to maintain complete program logic.

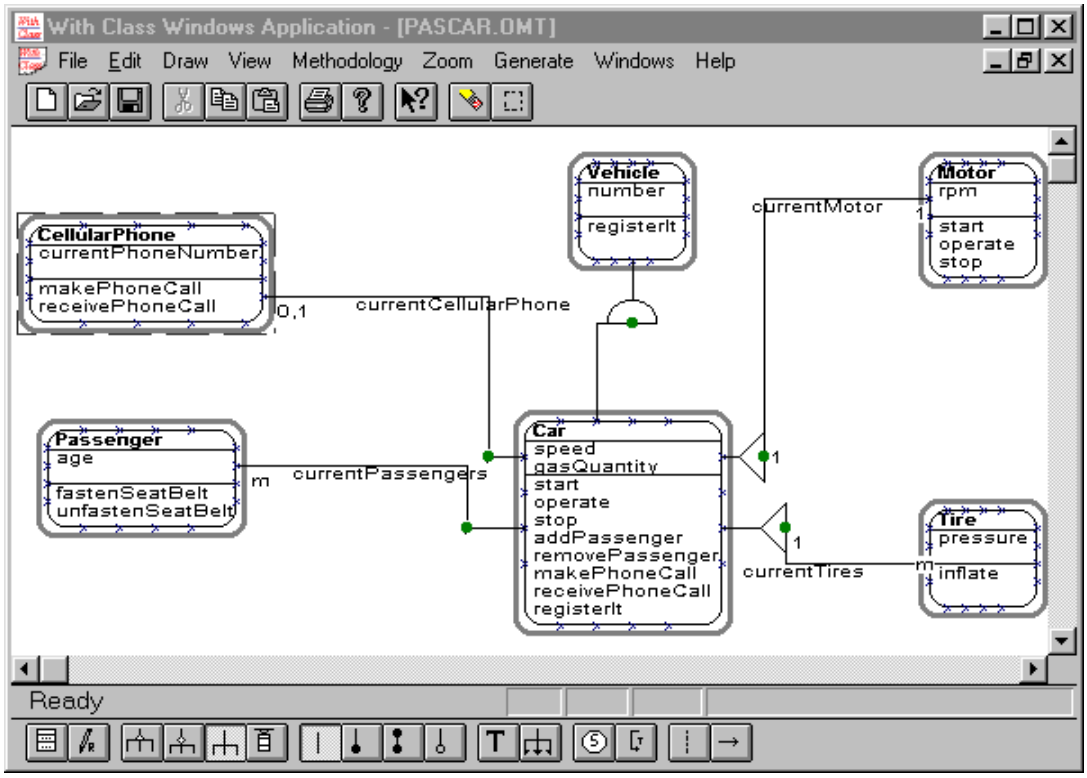

Figure 13 - With Class 2.5 using Coad-Yourdon Object Model

WithClass a CASE tool (Figure 13) currently fully supports the Delphi language to produce complete Delphi classes from OMT, Booch, Schlaer-Mellor, or Coad-Yourdon OO modeling. The model shown is using Coad-Yourdon, the code output for the passenger class is shown (Figure 14).

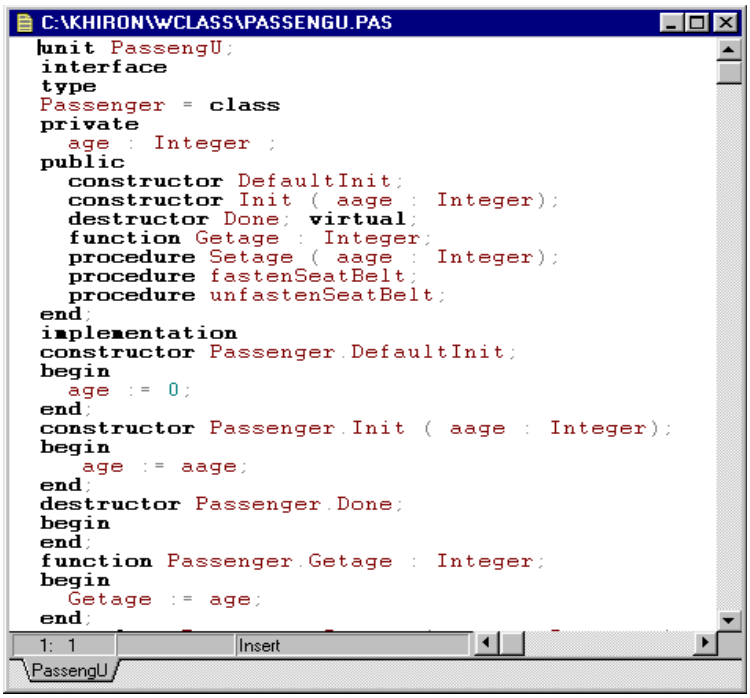

Figure 14 - OOAD Tools like WithClass 2.5 create Delphi Objects.

C Transaction Processing

Most SQL Servers provide a level of transaction control that allows the developer to guarantee that either an entire database transaction against the server completes, or is rolled back in it's entirety. Transaction Processing tools enable the developer to manage successful commit or rollback of entire transactions beyond the server to heterogeneous multi server environments and are very important going forward with multi-tier client/server environments.

Company	*Product*	*Description*
BEA Systems	Tuxedo	Delphi solutions for transaction processing.
Transarc	ENCINA	See Open Horizons *Connection* product.
Open Horizon	Connection	Provides a native connection to Transarc ENCINA transaction processing architecture (DCE) allowing Delphi applications to be deployed into a stable 3-tier architecture.
Other	Custom TP technology	Delphi's data classes have transaction support in-built, with hooks in the Database class to enable extensions to any custom transaction monitor.

Table 6 - Transaction Processing systems

ENCINA from Transarc is such a system, based on DCE (Distributed Computing Environment) RPC (Remote Procedure Calls) and it offers transactional control to any XA standard compliant device including RDBMS's, Mainframe applications (IDMS, ISAM, etc.) and hardware devices that support the

XA standard. Delphi can currently integrate with ENCINA through Open Horizon's Connection ODBC product, which enables Delphi to use the BDEs ODBC (Open DataBase Connectivity) interface.

Another product in this category is Tuxedo from BEA Systems, which is also XA compliant. It provides a similar transactional semantic, however it relies on a propriety call interface.

Additionally Delphi Client/Server Suite 2.0 has added a new Data Module Object type to facilitate the separation of application logic (business rules) from data presentation and data access.[5] This allows developers to logically partition applications into a multi-tiered design within the visual environment while improving reuse and maintenance. As Delphi can directly call RPC (Remote Procedure Call) interfaces, such as DCE or the forthcoming Network OLE, partitioning can be physically separated at any tier boundary. Delphi Client / Server Suite 2.0's support for OLE and remote automation also enables physical partitioning of business logic.

VII Summary

Delphi is a high performance visual client/server development tool for Windows. Backed by over 10 years of solid native code compiler experience, Delphi is the first product to combine a state of the art compiler with visual development and scaleable database technology in a robust object-oriented environment. Unlike any other development tool currently available, Delphi provides much-needed rapid development support for production-quality client/ server and Windows applications with an Open Tools API to integrate with third party technologies.

Over a quarter of a million developers world-wide are today using Delphi to produce a diverse range of solutions such as integrated applications on the Information superhighway, secure funds transfer systems, multi-media publishing, and line of business solutions by some of the world's leading IS departments.

Delphi customers include such organizations as Alcatel, Arthur Anderson, BMW, BP Shipping, Bank of America, BBC Television, British Telecom, City of Los Angeles, Conoco, DHL, Fiat, First National Bank of Chicago, KPMG, Mercury Communications, SwissBank SG Warburg, Unibank and many others. Delphi and Delphi Client/Server are being used for a broad range of applications that demand rapid application development and high performance.

With it's Open Tools API Borland fulfills its commitment to developers world-wide to provide state-of-the-art technology integrated tightly within the Delphi environment. By publishing comprehensive integration interfaces and utilizing industry standards like OLE and Intersolv PVCS Delphi provides developers with productive and powerful tools for the future. Delphi offers the fastest way to the fastest applications today, and a healthy future is assured with further development both by Borland, and third party tool vendors.

[5] See Delphi 2.0 Client Server Database Architecture white paper for more information.

VIII Technical Appendix

A Exporting classes in Delphi

Let's look at how Delphi passes fully functional Objects to DLL's and uses Objects from DLL's in it's internal functioning.

Firstly an ancestor class of the class you want to transfer is defined and will be the published interface available to both modules. This base class must define all the methods to be called across the DLL boundary. Each method must be defined as a Virtual function to allow a descendant class to override the function. Each method is also defined as an export function allowing special entry and exit code to be created for the function just as if were a standard DLL export clause. This class type is often called a pure virtual base class.

The DLL defines a function that returns an instance of our base class, the executable (in this case Delphi) calls that function, but the DLL actually passes back a pointer to it's functional subclass. The executable now proceeds to call the virtual function by the standard virtual mechanism, i.e. offsets into the VM Table passed in the first 4 bytes of the Object. However this is a VMT built in a totally different module with far calls directly into that modules implementations of the overridden functions, so the object code to call that routine calls a routine in the DLL. The only difficult bit in Delphi 1.0 is to protect Exceptions crossing the boundary. As is the case with normal DLL functions, stack unwinding across the boundary will cause serious problems, so it is imperative to put exception traps around all exported method implementations. This is not necessary in Delphi 2.0 as exceptions are raised as system exceptions and thus may pass the DLL boundary.

B File VIRTINTF.PAS

1 Function ReleaseException - exception protection

This function removes the current exception object from the exception stack and return it's message string, in Delphi 2.0 this function is kept for backward compatibility, but does not modify the exception stack.

2 Class tInterface - the ancestor of all interface classes

The first virtual exportable Class is used as a base type for all of the other classes, it is called tInterface. Its is a simple class that defines a function to report the descendants version and redefines the free method to call an exported virtual release method, thus allowing classes to be disposed externally. Delphi Client/Server Suite 2.0 adds the ability to maintain a reference count allowing the class to be referenced from many places and only destructed by the last location interested.

Method	*Version*	*Function*
function AddRef: Longint;	2.0	Increments and returns internal reference count
function Release: Longint;	1.0	Used to destruct remote class (Delphi 2.0 will decrement the refference count and dedestruct the last instance)
function GetVersion: Integer;	1.0	Return interface version (Default = 0 for Delphi 1.0, 2 for Delphi 2.0)

Table 7 - Virtual methods for tInterface

3 Class TIStream - File handling from External modules

This type is a descendant of Tinterface that defines an interface for passing Streams to external modules, the useable stream classes are in Istreams.pas. The TIStream defines the exported core stream input and output routines to talk to Interface streams declared in IStreams. OpenTools defines these to allow in memory streams of unsaved project files to be accessed from tool routines.

Method	*Version*	*Function*
function Read(var Buffer; Count: Longint): Longint;	1.0	Try to read Count characters into Buffer, and return the number of Characters actually read from the stream.
function Write(const Buffer; Count: Longint): Longint;	1.0	Try to write Count characters from Buffer, and return the number of Characters actually written to the stream.
function Seek(Offset: Longint; Origin: Word): Longint;	1.0	Seeks to a location in the stream. If Origin is 0, the new position is Offset (seek absolute). If Origin is 1, the new position is Position + Offset (seek relative). If Origin is 2, the new position is Size + Offset (seek absolute from end of data). Seek returns the new position, relative to the beginning of the stream.
function GetModifyTime: Longint;	1.0	Returns the files date and time (returns 0 when attached to a tIMemoryStream, when attached to a tIFileStream returns the date when a file was created or last modified in DOS internal format.)
procedure SetModifyTime(Time: Longint);	2.0	Sets the file date/time when attached to a tIFileStream.

Table 8 - Virtual methods for tIStream

C File ISTREAMS.PAS

The Istreams unit contains definitions for two subclasses the tiMemoryStream and the tiFileStream which contain a physical stream (tMemoryStream and tFileSTream respectively). These stream holders are exported across the DLL boundary to a module that passes them to the constructor of a tVirtualStream. The module that contains the tVirtualStream can then access the stream as if it were a local file or memory stream, the actual stream being in a foreign module.

1 Const ExceptionHandler: TExceptionHandler

This method pointer points to a local exception handler for handling streaming exceptions.

2 Class tIMemoryStream and tIFileStream - Stream exporters

These two subclasses are exported as the tiStream base class, into the using module where they are then passed in to the constructor of a tVirtualStream. The interface file stream is constructed by passing it a file name and the mode to open it. The interface memory stream is constructed by passing it a pointer to another Stream, or nil for the object to construct one from scratch and remove it when finished.

3 Class TVirtualStream - File handling from External modules

The tVirtualStream is a subclass of a tStream, and can be accessed in the calling Module.

Method	*Version*	*Function*
constructor Create(AIStream: TIStream);	1.0	Passed a tIStream will instruct all stream access routines to call across the interface into the stream interface exporting module.
function Read(var Buffer; Count: Longint): Longint;	1.0	Calls Read function of tIStream in foreign module.
function Write(const Buffer; Count: Longint): Longint;	1.0	Calls Write function of tIStream in foreign module.
function Seek(Offset: Longint; Origin: Word): Longint;	1.0	Calls Seek function of tIStream in foreign module.
function GetModifyTime: Longint;	1.0	Calls GetModifyTime function of tIStream in foreign module.
Procedure SetModifyTime(Time: Longint);	2.0	Calls SetModifyTime function of tIStream in foreign module.

Table 9 - Virtual methods for tVirtualStream

D File DSGNINTF.PAS - The Designer Interface

Dsgnintf defines a series of classes descending from tDesigner that are called by Delphi to edit component properties. The mechanism for creating a custom property designer is not trivial, and is covered in depth in the Delphi technical manuals, and well documented in the file DSGNINTF.PAS in the VCL source, so I shall show how the designers are exported from the component library COMPLIB.

A designer is created by subclassing the generic tPropertyEditor class, which defines a number of functions for setting and getting properties, defining sub-properties (i.e.: tFont), defining the style of editing (Dialogs, DropDown List, Simple, Read Only, and multi Component selectable). Once a tPropertyEditor subclass has been defined in a unit with the necessary information to define it's editing parameters, that unit is added to the Component Library through the Options|Install function.

There is one other exported definition needed by COMPLIB to create an instance of that editor upon request , it needs a procedure in the unit called Register (Called in the same way as the register routine for Components). That function has to call the function RegisterPropertyEditor passing the type of the Property, the Component Class or Descendants that the editor will apply to, the name of the property, and the Class of the Editor. The component library then creates a list indexed on the definitions to create the appropriate Editor whenever the Object Inspector requests an editor for a property.

E File TOOLINTF.PAS - the Tool Interface

The tiToolServices Object is one of the key exported objects in the OpenTools API, it is essentially a class exported from Delphi to external modules to allow them to take control of the Delphi environment. It is a subclass of the basic tInterface class, and contains exported virtual function to provide Project and File Management routines, Dialogs for non Delphi DLL's, a Component Library interface, and a routine to access Delphis standard exception handler. The Tool services object is created on the application side, and passed to the VCS/Expert Manager DLL during initialization. Note: The application is responsible for creating and freeing the interface object, and the client should never free the interface.

1 Class tToolServices - Actions

These routines allow the module passed the tIToolServices class to instruct the Delphi IDE to load and close files, projects, and forms.

Method	*Version*	*Function*
function CloseProject: Boolean;	1.0	returns True if no project open, or if the currently open project can be closed.
function OpenProject(const ProjName: string): Boolean;	1.0	returns True if the named project can be opened. Pass an empty string to create a new project and main form.
function OpenProjectInfo(const ProjName: string): Boolean;	1.0	returns True if the named project file can be opened. This routine bypasses all the normal project load features (such as loading a desktop file, showing the source code, etc), and simply opens the .DPJ and .OPT files.
function SaveProject: Boolean;	1.0	returns True if the project is unmodified, if there is no project open, or if the open project can be saved.
function CloseFile(const FileName: string): Boolean;	1.0	returns True if the specified file is not currently open, or if it can be closed.
function SaveFile(const FileName: string): Boolean;	1.0	returns True if the specified file is successfully saved as FileName.
function OpenFile(const FileName: string): Boolean;	1.0	returns True if the specified file is already open or can be opened.
function ReloadFile(const FileName: string): Boolean;	1.0	returns True if the file is already open and was reloaded from disk. (NOTE: This will not perform any checking of the current editor state).
Function ModalDialogBox(Instance: THandle; TemplateName: PChar; WndParent: HWnd; DialogFunc: TFarProc; InitParam: LongInt): Integer;	1.0	used by non-VCL DLL's to present a dialog box which is modal. Note that DLLs written using VCL can simply call a form's ShowModal function.
function CreateModule(const ModuleName: string;Source, Form: TIStream; CreateFlags: TcreateModuleFlags): Boolean;	1.0	Will create new module from memory images of the source and, optionally, the form file. The CreateModuleFlags are: **cmAddToProject** Add the new module to the currently open project. **CmShowSource** Show the source file in the top-most editor window. **CmShowForm** If a form is created, show it above the source. **CmUnNamed** Will mark the module as unnamed which will cause the SaveAs dialog to show the first time the user attempts to save the file. **CmNewUnit** Creates a new unit and adds it to the current project. NOTE: all other parameters are ignored. **CmNewForm** Creates a new form and adds it to the current project. NOTE: all other parameters are ignored. **CmNewModel** Creates a new Data Model and adds it to the current project. NOTE: all other parameters are ignored. **CmMainForm** If the module includes a form, make it the main form of the currently open project. Only valid with the cmAddToProject option. **CmMarkModified** Will insure that the new module is marked as modified. **CmExisting** Will Create a module from an existing file on disk
function CreateModuleEx(const ModuleName, FormName, AncestorClass,FileSystem: string; Source, Form: TIStream;CreateFlags: TCreateModuleFlags): TIModuleInterface;	2.0	New extended form of CreateModule. This will return a TIModuleInterface. All CreateModes from CreateModule are supported with only the following differences: **cmExisting** Will create an existing module from the given file system. **AncestorClass** This must specify an existing base class in the project (use the cmAddToProject flag to add a module to the project first). NOTES: Pass an empty string for the file system parameter in order to use the default file system. The file system parameter *must* be a valid file system previously registered through the RegisiterFilesystem API.

Table 10 - Virtual Action methods for tIToolServices.

2 Class tToolServices - Informational Services

These routines allow the module passed the tIToolServices class to query the Delphi IDE about the state of the projects, files and forms it has loaded.

Method	*Version*	*Function*
function GetParentHandle: HWND;	1.0	returns a HWND, which should be used as the parent for any windows created by the client.

function GetProjectName: string;	1.0	returns a fully qualified path name of the currently open project file, or an empty string if no project is open.
function GetUnitCount: Integer;	1.0	returns the current number of units belonging to the project.
function GetUnitName(Index: Integer): string;	1.0	returns a fully qualified name of the specified unit.
function GetFormCount: Integer;	1.0	returns the current number of forms belonging to the project.
function GetFormName(Index: Integer): string;	1.0	returns a fully qualified name of the specified form file.
function GetCurrentFile: string;	1.0	returns a fully qualified name of the current file, which could either be a form or unit (.PAS). Returns a blank string if no file is currently selected.
function IsFileOpen(const FileName: string): Boolean;	1.0	returns True if the named file is currently open.
Function GetNewModuleName(var UnitIdent, FileName: string): Boolean;	1.0	Automatically generates a valid Filename and Unit identifier. Uses the same mechanism as used by the IDE.

Table 11 - Virtual Information methods for tIToolServices.

3 Class tToolServices - Component Library interface

These routines allow the module passed the tIToolServices class to query the Delphi IDE about the Component library, the component installed, and the palettes they are in.

Method	*Version*	*Function*
function GetModuleCount: Integer;	1.0	Returns the number of currently installed modules in the component library.
Function GetModuleName(Index: Integer): string;	1.0	Returns then name of the module given its index.
Function GetComponentCount(ModIndex: Integer): Integer;	1.0	Returns the number of components installed in a particular module.
Function GetComponentName(ModIndex, CompIndex: Integer): string;	1.0	Returns the name of the component given its module index and index in that module.

Table 12 - Virtual Component library methods for tIToolServices.

4 Class tToolServices - Virtual File System interfaces

These routines allow the module passed the tIToolServices class to install a custom filing system, to allow for example Blob Storage of all project files for complex version control systems.

Method	*Version*	*Function*
Function RegisterFileSystem(AVirtualFileSystem: TIVirtualFileSystem): Boolean;	2.0	Registers an externally defined file system.
Function UnRegisterFileSystem(const Ident: string): Boolean;	2.0	UnRegisters an externally defined file system.
Function GetFileSystem(const Ident: string): TIVirtualFileSystem;	2.0	Returns an installed external file system based on the identifier passed.

Table 13 - Virtual File system methods for tIToolServices.

F File VCSIntf.PAS - The Version Control Interface

This file defines the cooperative interface between the Delphi IDE and a VCS Manager DLL. The VCS Manager DLL must be named in the DELPHI.INI file at startup time under the isVersionContol section, as the value for the ivVCSManager variable (Defined as VCSManager). e.g.

```
[Version Control]
VCSManager=C:\DELPHI\TOOLS\RAM_VC.DLL
```

When the IDE loads, it will load the specified DLL and attempt to obtain an address for the DLL's initialization function, exported by a TVCSManagerInitProc function named by VCSManagerEntryPoint. Delphi is responsible for freeing the client object before unloading the VCS Manager DLL.

1 Function TVCSManagerInitProc - Interface builder function

A function matching this signature must be exported from the VCS Manager DLL.
tVCSManagerInitProc = function (VCSInterface: TIToolServices): TIVCSClient; Note that in this function the Version Control manager is passed a ToolServices interface from within Delphi so that the Version control system has full access to all modules within the Delphi IDE.

2 Class tIVCSClient - Version control

tiVCSClient has functions to create Menu items within the Delphi menu, respond to those items being activated, check and disable menu items, and finally a Project Change notification interface so that Version control software encapsulated by the Class can respond to new projects.

Method	*Version*	*Function*
function GetIDString: string;	1.0	Called at initialization. Client should return a unique identification string. 'Borland.StdVcs' is reserved for Borland use.
Procedure ExecuteVerb(Index: Integer);	1.0	Called when the user selects a verb from a menu.
function GetMenuName: string;	1.0	Called to retrieve the name of the main menu item to be added to the application's menu bar. Return a blank string to indicate no menu.
function GetVerb(Index: Integer): string;	1.0	Called to retrieve the menu text for each verb. A verb may be returned as a blank string to create a seperator bar.
function GetVerbCount: Integer;	1.0	Called to determine the number of available verbs. This function will not be called if the GetMenuName function returns a blank string (indicating no menu).
function GetVerbState(Index: Integer): Word;	1.0	Called to determine the state of a particular verb. The return value is a bit field of various states. **VsEnabled** Verb enabled if set, otherwise disabled **vsChecked** Verb checked if set, otherwise cleared
procedure ProjectChange;	1.0	Called when there is any state change of the current project, i.e. when a project is destroyed or created.

Table 14 - Virtual methods for tIVCSClient.

G File FileIntf.PAS - The Virtual File System

This is the definition of the IDE's virtual file system. An expert, VCS manager, property editor, or component editor can register a new file system interface with the IDE. This allows for re-vectoring of file operations to the editor and form/data model designer. The way to specify an alternate file system for a file, is to open it through the Tools API (ToolIntf.pas). The default file system will always be used by the IDE except in the case mentioned above.

1 Class TIVirtualFileSystem - Virtual storage class

All references to the term "file" depend on how it is defined by the file system. The "file" could be a Memo Blob field, SQL text, etc... A file system instance must provide the following;

Method	*Version*	*Function*
function GetFileStream(const FileName: TFileName; Mode: Integer): TIStream;	2.0	This is the core of the file system. The file system must return an instance of a TIStream for reading/writing according to the Mode.
function FileAge(const FileName: TFileName): Longint;	2.0	This should return long value corresponding to the DOS file date format.
function RenameFile(const OldName, NewName: TFileName): Boolean;	2.0	Returns True if the file system was able to rename the file.
function IsReadonly(const FileName: TFileName): Boolean;	2.0	Return True if the given file is read only.
function IsFileBased: Boolean; virtual; export; abstract;	2.0	Return True if the file system closely matches the OS file system. If this is False, certain operations are not performed.
function DeleteFile(const FileName: TFileName): Boolean;	2.0	Return True is the file was successfully deleted.
function FileExists(const FileName: TFileName): Boolean;	2.0	Return True if the specified file exists in this file system.
function GetTempFileName(const FileName: TFileName): TFileName;	2.0	Returns a Temporary file name based on the name given.
function GetBackupFileName(const FileName: TFileName): TFileName;	2.0	Returns a backup file name based on the name given. By convention, the extension is shifted one character to the right and a tilde '~' character is inserted. (eg unit1.pas -> unit1.~pa).
function GetIDString: string;	2.0	Returns a unique ID string used to identify the file system. By conventions this string should be in the form <Vendor or Product>.<FileSystemName>. (eg. Borland.SQLFileSystem).

Table 15 - Virtual methods for TIVirtualFileSystem.

H File ExptIntf.PAS - The Expert Interface

The final class interface is the expert interface, which is documented by example in the ExptDemo application shipping with Delphi Client/Server samples. An expert can be loaded in two ways, in an external DLL module using a similar interface to the VCS interface, or as a Library Expert. The advantage of the later method is that an expert compiled into the COMPLIB has complete access to instances of all registered classes, making it possible to create complex experts.

Method	*Version*	*Function*
function GetName: string;	1.0	REQUIRED. This must return a unique descriptive name identifying this expert.
function GetComment: string;	1.0	REQUIRED if style is esForm or es Project. This should return a 1 - 2 sentence describing the function of this expert.
function GetGlyph: HBITMAP;	1.0	REQUIRED if style is esForm or esProject. This should return a handle to a bitmap to be displayed in the form or project list boxes or dialogs. This bitmap should be 60x40 pixels.
function GetStyle: TExpertStyle;	1.0	REQUIRED. Returns one of three possible values: **esStandard** Tells the IDE to treat the interface to this expert as a menu item on the Help menu. **EsForm** Tells the IDE to treat this expert interface in a fashion similar to form templates.

		EsProject Tells the IDE to treat this interface in a fashion similar to project templates.
function GetState: TExpertState;	1.0	REQUIRED. If the style is esStandard, esChecked will cause the menu to display a checkmark. NOTE: This function is called each time the expert is shown in a menu or listbox in order to determine how it should be displayed.
function GetIDString: string;	1.0	REQUIRED. This ID string should be unique to all experts that could be installed. By convention, the format of the string is: CompanyName.ExpertFunction, eg. Borland.WidgetExpert
function GetMenuText: string;	1.0	REQURED if style is esStandard. This should return the actual text to display for the menu item. NOTE: This function is called each time the parent menu is pulled-down, so it is possible to provide context sensitive text.
function GetMenuText: string;	1.0	REQUIRED. Called whenever this expert is invoked via the menu, form gallery dialog, or project gallery dialog. The style will determine how the expert was invoked.

Table 16 - Virtual methods for tIExpert.

You can load an expert as a DLL by registering it in the Delphi.ini file or Registry (Window 3.x and Windows95 respectively) in the experts section;

```
[Experts]
ExptDemo=D:\BORLAND\DELPHI\BIN\EXPTDEMO.DLL
```

You will need to make a DLL that exports a function with a name defined in ExpertEntryPoint ('INITEXPERT0012') of the prototype TExpertInitProc = function(ToolServices: TIToolServices; RegisterProc: TExpertRegisterProc; var Terminate: TExpertTerminateProc): Boolean; Note that the Function is passed the same TToolServices class passed to a Version Control Manager DLL. It is also passed a Registration and Termination routines. The Registration routine is passed a constructed Expert which will be a subclass of TIExpert. A Library Expert is instantiated in COMPLIB and registered through the RegisterLibraryExpert function in a similar manner to the registration of a component. The expert has access to tToolServices from an instance in the ExptIntf unit.

IX Bibliography

'The Delphi Open Tools Interface'
Richard Morris, KHIRON Software
Borland Developers Conference, SAN DIEGO August 1995

'Team Development with Delphi'
Brian Slatner, Turbo Power Software
Borland Developers Conference, SAN DIEGO August 1995

'An Overview of the 32 Bit Delphi Compiler for Win 95 and NT'
Zack Urlocker, Borland International
Borland Developers Conference, SAN DIEGO August 1995

'Rapid Application Development with Delphi'
Diane Rogers, Borland International
Borland Developers Conference, SAN DIEGO August 1995