A Comparison of Clarion for Windows to Delphi

By Bruce D. Barrington
Chairman, TopSpeed Corporation

Clarion for Windows from TopSpeed Corporation and Delphi from Borland International are both leading edge database development tools launched within the same year. These tools have more in common than their age and purpose, however. Clarion and Delphi share ancestry.

Delphi is the latest evolution of Pascal for Windows which evolved from Turbo Pascal, Borlands first language product. Five years after Turbo Pascal appeared, Niels Jensen, Borlands founder, and the language development team left Borland to found Jensen and Partners, International (JPI). JPI purchased their work in progress from Borland which became the TopSpeed line of optimizing compilers.

Four years later, JPI merged with Clarion Software to form TopSpeed Corporation. Clarion for Windows was created by merging the Clarion application generation and database technology with the TopSpeed compiler technology. While Clarion for Windows and Delphi both produce database applications, they go about it in very different ways. The following comments compare Clarion for Windows Version 1.5 to Delphi Version 1.1:

Application Generator

The primary enabling technology in Clarion is a powerful visual template language that controls a high speed source code generating engine. The template language also manages the development environment and provides a user interface for gathering specifications defining application behavior. Visual templates substantially increase development productivity by generating source code that otherwise must be written by hand. Because the Clarion template language is an open technology, the application generator can be continuously augmented with third-party or user written templates.

The application repository can be customized by embedding source code to produce alternative or additional behavior. As a result, applications are continuously maintained as repositories, extending the productivity benefits to their entire useful life.

Delphi uses Database Form Experts to generate database update forms into an application. Form experts are one-time processes. In other words, any changes to the source code will be lost if the expert is reused.

Application Model

The Clarion application model treats an application as a single entity rather than a disconnected set of forms and/or procedures. An application model consists of application properties along with procedure definitions that are related logically (as calling sequences) and physically (in source modules).

Procedure definitions contain formats of the visual elements such as menus, windows, and reports. In addition, procedure definitions contain procedure and control properties that specify the behavior required of the procedure.

The application properties and procedure definitions are maintained in an application repository. When combined with a data dictionary, the application repository can be manufactured into a complete, functioning application with a single mouse click.

Delphi has no application model, application repository, or data dictionary. Delphi applications are defined by project files that list the forms and source units to be processed by the compiler.

Application Tree

The Clarion application model is maintained in a persistent, reconfigurable, collapsible, expandable tree structure. Tab controls are used to reconfigure the tree into four different views: The module view groups procedures into source modules. The procedure view illustrates the calling sequence (procedures branch from their caller). The template view groups procedures by template type. And the procedure name view displays procedures alphabetically. Procedure names are displayed along with their template type and description.

The application tree is the control center for application development. The right mouse button produces a pop-up menu to invoke the window formatter, report formatter, local data editor, embedded source code editor, or template property dialogs for any procedure. Large projects are managed by compressing the tree revealing only details that are currently relevant. Importantly, the configuration of the application tree persists across design sessions.

Delphi applications are maintained by a project manager dialog listing the filenames and paths of the units (.PAS files) and forms (.DFM files) that comprise the project. Delphi makes no provision for representing applications graphically, describing component files, or managing large projects.

Application Wizard

The Clarion application wizard creates a fully functional multi-threaded MDI application from a data dictionary. The MDI frame contains standard File, Edit, and Help menus along with menus for browsing and printing files. Browse/form dialogs and   report procedures are created for every file in the dictionary. Browse dialogs feature tab controls that select the key sequence used to display records. Form dialogs for parent files in a one-to-many relationship feature tab controls that browse selected child records.

The application wizard fills out the necessary application and procedure properties and populates windows and reports. The resulting application can be modified and enhanced using standard Clarion visual tools. Applications produced by the wizard are indistinguishable from a developers own work.

Delphi has no similar functionality.

Migrating Legacy Code

The Clarion application wizard enables a convenient three step migration strategy for producing high quality Windows applications from legacy data: First, the data file structures are imported into a Clarion data dictionary. Standard Clarion database drivers provide this service for dBase, Clipper, FoxPro, Btrieve, and Clarion files. The ODBC database driver can be used for most other data formats. Next, the file relationships are added. Finally, the application wizard creates the application.

Fussing over the dictionary by correcting prompting phrases and column headings, adding descriptions and pictures, disabling unneeded keys, etc. substantially improves the quality of the resulting application.

All features of the legacy application may not be reproduced using this strategy, however, the new application is fully capable of maintaining the legacy data in place concurrently with the legacy application. In addition, the new application features an attractive Windows 95 style MDI user interface--a result that would be impossible using a code conversion strategy.

Delphi offers no special strategy for migrating legacy applications.

Procedure Generation

Clarion offers a comprehensive library of procedure templates that automatically generate source code for a wide variety of menu, browse, form, report, batch, and other types of procedures. Generated procedures can be modified and enhanced by embedding custom source code. This way, procedures can be maintained in template form for their entire useful life, substantially reducing the effort required for documentation, maintenance, and testing.

Wizards are provided for the Browse, Form, and Report templates that automatically fill out procedure and control properties and populate windows and reports. The Browse and Form wizards automatically produce additional Browse and Form procedures for related files. Procedures produced by these wizards are indistinguishable from the developers own work.

Delphi provides Database Form Experts that produce source code and populate windows for accessing and updating databases. These experts offer horizontal, vertical, or spreadsheet orientation of the form window. After a Delphi procedure is generated by an expert, it must be forever maintained as source code because hand-coded modifications are not recognized by the Form Expert in future sessions. Delphi does not provide experts for creating browse or report procedures.

Selective Inheritance

Clarion packages reusable code in the form of templates which generate made to order source code. Features that are not requested in the design do not exist in the generated code. As a result, template authors can include any feature they can imagine without producing unwanted overhead in the resulting application.

Delphi reusable code comes in the form of objects. To reuse any part of an object, the object must be inherited in its entirety. Swallowing objects whole when only a nibble would suffice is the primary cause of the problem known as object bloat (the tendency of object-oriented languages to produce huge executables.).

Feature Grafting

A Clarion extension template can be used to graft features onto another template without modifying or inheriting it. The target template remains encapsulated so that it can be changed without any knowledge of its grafted features. Extension templates that graft to the same target template but were obtained from different sources live harmoniously without any knowledge about one another. All extension templates, no matter when they were grafted or in what order, can be used with the target template.

Under the object model used by Delphi, a class must be inherited in order to be extended. Accordingly, new features must be prioritized and packaged like a stack of Russian dolls--each feature encapsulating all prior features. Features of objects derived from the same base class are encapsulated against one another and cannot be shared.

Incremental Testing

The Clarion application generator temporarily resolves forward procedure references with empty ToDo procedures permitting incremental testing. In order to test an application, a Delphi programmer must prototype and implement every referenced procedure.

Multiple Document Interface (MDI)

Clarion supports the Multiple Document Interface at all levels. The application wizard produces MDI applications by default. A Browse procedure is produced for every key defined in the underlying file. These procedures are placed in separate MDI Windows using separate threads. As a result, multiple Browses can be active at the same time even when they access the same file. All Clarion procedure templates offer the option of running in an MDI child window under a separate thread. Delphi supports MDI only at the lowest level.

RelationTree Control

A Clarion list box can display an internal memory queue in the form of an outline. Standard plus and minus icons control the process of expanding and collapsing the outline. Each level of the outline can be associated with a unique file icon and can be displayed in a unique color.

The RelationTree control template generates a list box that displays nested one-to-many relations as a tree of records. A record that owns child records is marked by a plus icon. Clicking on the plus icon expands the tree by displaying the child records while changing to the minus icon. Clicking on the minus icon collapses the tree by hiding the child records while changing to the plus icon. Each file can be associated with its own file icon, display color, and update procedure.

Consider three files: CUSTOMER, INVOICE, and ITEM. The CUSTOMER file contains a record for every customer. The INVOICE file contains a record for every order placed by every customer. And the ITEM file contains a record for every line item in every order. Initially, a RelationTree list box displays only customer records. Every customer with orders is displayed with a plus icon. Clicking the plus icon displays that customers orders. Clicking the plus icon on an order displays its line items. The right mouse button pops-up an Insert/Change/Delete menu that invokes an appropriate update procedure for the selected record.

Delphi does not have an outline control.

Data Dictionary

The Clarion Data Dictionary serves as a repository for database information such as database drivers, connection instructions, access keys, record layouts, file relationships, relational integrity constraints, data validity rules, display pictures, and visual field representations. Delphi has no data dictionary.

Window Controls

There are many ways to represent data visually in windows and reports. A window control requires a descriptive prompt label optionally containing a hot letter accelerator. The control may require a tool tip (balloon help) or a description in the status bar. The control itself is an option. For example, a string field with a fixed set of valid values can be represented by a group of radio buttons, a list box, or a drop down list box. Numeric, date, and time fields require input masks and display pictures. The Clarion data dictionary is the repository for these visual design elements. Populating windows and reports from the data dictionary saves design time and produces visual consistency. Since Delphi has no data dictionary, all visual design elements must be re-entered for every occurrence of a field.

Persistent Relationships

The process of database design involves defining tables and their relationships. A many-to-one relationship simply reduces the storage required by eliminating duplicate parent data contained in child records. When a child record is accessed, data from the parent record is likely to be accessed also. The Clarion data dictionary serves as a repository for these relationships. When populating a window or report from a table, all fields from related tables are also available for use. Templates automatically generate the necessary code to look up secondary data Since Delphi has no data dictionary, link fields defining the relationship must be re-entered every time tables are joined.

Referential Integrity

The Clarion data dictionary offers standard referential integrity constraints to control one-to-many relationships. The Form template automatically generates the code necessary to implement these rules in Clarion applications. Even if referential integrity rules have been implemented on a database server, using triggers and stored procedures, a good client application avoids such errors by providing its users with an early warning. Delphi has no data dictionary and offers no technology for implementing referential integrity constraints.

Application Maintenance

A Clarion application repository maintains links into its underlying data dictionary. Changes to the data dictionary are automatically reflected in an application every time it is generated. When a Delphi database changes, Delphi applications must be completely reexamined, manually correcting every instance of every changed element.

Threaded File Access

Clarion database drivers provide safe concurrent database access from multiple threads within an application. When a file is opened on a new thread, a new instance of its file control block and record buffer are created. Concurrency within an application is then controlled using the same process that provides concurrency protection from other applications. Such functionality is necessary to produce MDI database applications. Delphi makes no provision for multi-threaded database access.

Btrieve Support

The Clarion Btrieve database driver provides native Btrieve file access using the standard Clarion database access grammar. The Btrieve driver supports both local and client/server versions of Btrieve as well as many of the extended Btrieve functions. Delphi does not support native Btrieve access.

Language Features

Clarion for Windows uses Clarion, a fourth generation language designed specifically for desktop business applications. Clarions built-in abstractions for standard business objects such as windows, reports, files, views, and queues produce source code that is easy to write, easy to read, and easy to learn. The Clarion language is named for its dictionary definition:

clarion *adj*. brilliantly clear

Delphi uses Object Pascal, an object oriented derivative of Turbo Pascal along with the Visual Component Library, a class library that produces all windows behavior.

Statement Punctuation

Clarion statements coded on a single line require no punctuation. Clarion compound statements are terminated with a period or END keyword. Clarion assignment statements use the equal sign. All Delphi statements must be separated with semi-colons. Compound statements must be enclosed within begin...end keywords. Delphi assignment statements require the double key sequence :=.

Picture Strings

Clarion strings optionally contain pictures which automatically format and deformat the values assigned to and received from the string. Delphi offers no similar functionality and requires string formatting functions to produce formatted strings.

Decimal Data Type

Delphi does not offer a packed decimal data type. Fractional numbers must be stored as a 4,6,8, or 10 byte floating point values. Clarion offers two packed decimal formats containing 1 to 31 decimal digits.

BCD Arithmetic

Clarion numeric expressions that do not contain floating point operands are evaluated using BCD (binary coded decimal) fixed point arithmetic. This process produces exact results so that a numeric expression such as:

3 +(1/3) - (10/3)

will evaluate **exactly** to 0. Delphi real expressions use floating point arithmetic which often introduces errors into financial calculations.

Mixed Expressions

Clarions safe typing permits string, numeric, date, and time data types to be freely mixed in a numeric and string expressions. Proper data conversions are automatically generated by the compiler. This design permits assignment statements such as:

```
Pi = (6 / 2)   & ( Circum / Diam / 2   - 3 )
```

The Delphi programmer must remember the data types of all variables and, if necessary, explicitly typecast them to the data type of the expression.

Foreign Data Types

In order to process data wherever it may exist, Clarion supports a comprehensive set of data types. In addition to standard integer and real data types, Clarion supports the following foreign data types which frequently occur in legacy databases:

Clarion packed decimal
IBM/EBCDIC packed decimal
Microsoft Basic 4 byte floating point
Microsoft Basic 8 byte floating point
Fixed length string
Null terminated (C style) string
Length prefixed (Pascal style) string
Btrieve 4 byte date
Btrieve 4 byte time

In addition to the standard integer and real data types, Delphi supports 6 and 10 byte reals and an 8 byte unsigned integer. Delphi does not support any foreign data types.

Polymorphic Parameters

Clarion supports polymorphic procedure and function parameters. As a result, Clarion functions and procedures can be written to accept parameters of any non-structured data type. Polymorphic parameters passed by value are automatically converted in the calling sequence generated by the compiler. Polymorphic parameters passed by address are converted during expression evaluation. The following Clarion program demonstrates the benefit of this functionality:

```
        MAP
            Divide3(*?)  !a polymorphic parameter (?)
        END              !passed by address (*)
A       LONG
B       DECIMAL(10,2)
C       REAL

        CODE
        Divide3(A)
        Divide3(B)
        Divide3(C)
        RETURN

Divide3 PROCEDURE(Param)
```

```
CODE
Param = 10 / 3
TYPE(Param)
RETURN
```

The values displayed by this program are 3, 3.33, and 3.33333333333333. This functionality is part of Clarions safe typing, a far more useful and forgiving strategy than strong typing used by Delphi.

String Slicing

A three character substring starting with the second character of a Clarion string can specified as String[2:4] (meaning characters 2 through 4 of the string). Such a string slice can be used on either the left or right side of an assignment statement. The Clarion compiler generates code that processes string slices in place without the overhead produced by sub-string functions. Delphi uses less efficient sub-string functions which cannot be used as the destination of an assignment statement.

Reference Variables

Clarion reference variables provide a safe reference to another data declaration. Reference variables are declared by pre-pending an ampersand to the target data type (e.g. &BYTE or &FILE). Reference variables are automatically dereferenced whenever they are used. Delphi uses pointer variables for this purpose which must be explicitly dereferenced.

Memory Queues

Clarion offers built-in support for memory queues. A Clarion QUEUE structure declares a record layout to be stored in memory (on the heap). Queue elements can be added, deleted, or retrieved either randomly and sequentially. When a Clarion queue grows to a size that will no longer fit in memory, it transparently spills into a designated disk file. Delphi has no similar functionality.

Structure Piercing

The Clarion deep assignment statement moves matching components from one structure to another as illustrated in the following example:

```
G1      GROUP,PRE(GP1)
A               BYTE
B               SHORT
        END
G2      GROUP,PRE(GP2)
A               BYTE
B               SHORT
C               LONG
        END
        CODE
        G2 :=: G1       !Is equivalent to       GP2:A = GP1:A
                        !                       GP2:B = GP1:B
```

The Clarion CLEAR procedure initializes every component of any structure or array to the proper binary representation for zero, blank, low value, or high value. As a result, an entire structure or array can be cleared, set to low values, or set to high values with a single statement. Delphi has no similar functionality.

Multi-Threading

A Clarion application may contain multiple asynchronous execution threads operating concurrently. These threads can be used for many purposes, such as: implementing a multi-document interface (MDI), preparing and printing reports in the background, filling read ahead buffers, etc. Clarion also supports a THREAD storage class for producing variables with a separate instance for each thread. Delphi cannot produce multi-threaded applications.

Run-time Expressions

The Clarion EVALUATE function evaluates a logical, numeric, or string expression contained in a string that can be modified at run-time. This functionality can be used for: user defined record filters, run-time formula entry, etc. Delphi has no similar functionality.

International Support

Clarion provides an environment file (*appname*.ENV) that contains information necessary to automatically localize any Clarion application. .ENV settings control the character set, collating sequence, date and time formats, upper/lower case pairs, standard button text, and standard error message text. Clarion also provides a comprehensive set of localization functions and international pictures. Delphi applications must be localized by explicitly coding string handling functions and by modifying the .EXE file with a resource editor.

Messaging Model

The Clarion messaging model is a simple ACCEPT...END loop which cycles when an event occurs. EVENT( ), FIELD( ), and FOCUS( ) functions supply the event that occurred, the Window control involved (if any), and the field that currently has focus. The events are typically processed by CASE structures. Adding a new event processing routine simply requires an additional OF ... clause--about the same overhead required by a single operand IF statement.

A Delphi program is primarily comprised of event processing procedures that are called by Windows. Each new event requires a new procedure prototype and a new procedure statement. Procedures are relatively costly ingredients containing overhead necessary to save and restore registers, create and destroy stack frames, etc.

Database Access

Clarion uses a database neutral grammar for declaring and accessing files and views of files from a database. As a result, the compiler verifies proper construction of files, views, and keys and flags invalid field names. All fields in a file or view are automatically retrieved with a single record access statement.

The Clarion database grammar assumes that every database possesses a powerful set of features including relational join, filter, and project operations as well as scrollable cursors. Clarion database drivers exploit the functionality available from the underlying database engine and supplement whatever is missing. As a result, Clarion applications are automatically optimized for all databases.

Delphi has no database access grammar. Delphi accesses databases through the Borland Database Engine (BDE) using *Tdatabase, Ttable,* and *Tquery* objects. Relational join, filter, and project operations are produced by moving a SELECT string to the *SQL* property of *Tquery*. Because SQL strings are not processed by the Delphi compiler, database related programming mistakes generate run-time errors.

Report Generation

Clarion includes a comprehensive report grammar that supports print preview, graphic forms, page headers and footers, group breaks, totaling and sub-totaling, and much more. Standard reports are normal components of a Clarion application.

Delphi does not provide report objects. All reports accompanying a Delphi application must be developed in and printed by a third party report writer such as Borlands ReportSmith.

Object Orientation

Clarion has no provision for declaring or deriving object classes. However, Clarion windows, reports, files, and views are proper objects with properties that are visible and changeable at run-time. Delphis Object Pascal is a thorough implementation of a single inheritance object oriented language.

Project System

The application generator automatically creates a TopSpeed project file which lists the source files, object modules, and DLLs necessary to make the target file. The TopSpeed project system then processes the project file by compiling source, as necessary, and linking the objects into an .EXE, .DLL, or .LIB. A windows resource file is produced as a by product of this process and is appended to the resulting file.

A Delphi project contains a list of Delphi source files (.PAS), Delphi object files, and Delphi form (.DFM) files. The Delphi project system compiles and links the source into an .EXE file and processes the form files to produce a windows resource file.

Mixed Language Projects

DLLs for the TopSpeed Modula-2 and C/C++ compilers can be purchased separately as Clarion add-ons. The project system examines the extension of each source file to determine which compiler to call. Clarion source files have an extension of .CLW. C and C++ source files have extensions of .C and .CPP, respectively. And Module-2 source files have an extension of .M2. (Include files, such as C headers can have any extension since they are invoked from standard source files.) Since all TopSpeed compilers share the same optimizing code generator, their object modules can be linked together seamlessly. As a result, any components of a Clarion application can be written in C, C++, or Modula-2.

All source units in a Delphi project must be written in Pascal.

Smart Method Linking

Smart Method Linking is a proprietary TopSpeed technology that eliminates unused virtual methods from executables. Since the Clarion run-time library is class based, Clarion executables only contain the library methods actually used. All virtual methods from the source code and the included library units are present in Delphi executables.

Register Parameters

The Clarion compiler generates register-based parameter passing when calling internal library and user-defined procedures and functions. This technique reduces stack overhead, increases performance, and decreases code size. Delphi requires stack-based parameters.

Standard Components

Both Clarion and Delphi produce reusable OBJ, LIB, and DLL files. Both products can use standard Windows DLL files produced by other development tools. Both products permit full access to the standard Windows API.

One-piece Executables

A Clarion application can be packaged as a single executable or split into an executable and a set of DLLs. The Clarion run-time library (which includes the print engine) and database drivers can be linked into the executable, one of the Clarion DLLs, or deployed as stand-alone DLLs.

Delphi applications can also be packaged into a single executable or an executable with DLLs. However, if the application accesses a database or prints a report, the Borland Database Engine and ReportSmith must be deployed separately. The Borland Database Engine consists of a minimum of 5 DLLs. SQL links average another 12 DLLs. The minimum run-time installation of ReportSmith produces 56 DLLs. As a result a typical Delphi database application consists of more than 60 EXE and DLL files.

Application Deployment

A typical Clarion database application that prints reports can deployed as a single executable of 800K or less. If multiple Clarion applications are deployed, the database drivers and run-time can be packaged separately. In this case, Clarion applications are typically less than 200K. The run-time library takes 644K (512K for the 32 bit run-time library). Clarion database drivers range in size from 20K for the ASCII driver to 160K for the dBase IV driver. The most popular drivers (Clarion, TopSpeed, and Btrieve) take about 50K each. Most Clarion applications can be stored on a single floppy disk.

A similar Delphi application produces a 400K executable. To support database access, however, the Borland Database Engine must be installed on the target machine. This installation requires two floppy disks and produces a directory containing 15 files totaling 2.44MB in size. To print reports, the ReportSmith run-time must also be installed on the target machine. A minimum installation (for dBase access only) requires 5 floppy disks and produces a directory containing 56 files totaling 5.19MB in size. In other words, deploying a typical Delphi application requires 8 floppy disks and occupies at least 8 megabytes on the target hard disk.

32 Bit Support

The Clarion IDE and Clarion applications are word size neutral. As a result, Clarion runs on all PC versions of Windows and produces native applications for all PC versions of Windows. The components of the IDE are 16 bit except for the 32 bit versions of the compiler, linker, and debugger which are, themselves, 32 bit components. For this reason, Clarion cannot make 32 bit applications while running under a 16 bit version of Windows.

By cloning 16 bit versions of Windows 95 components such as Property Sheets (tabbed dialogs) and Tool Tips (yellow balloon help), and by cloning a 32 bit version of VBX control support, Clarion produces both 16 and 32 bit Windows applications from the same source code and VBX controls. This strategy permits Clarion developers to provide new 32 bit versions of their applications while continuing to support identical 16 bit versions.

As of this writing, Delphi has no 32-bit support.

Platform Transition Strategy

PC developers and their end-users run four different versions of Windows: Windows 3.x, Windows 3.x under OS/2, Windows 95, and Windows NT. Windows 3.1 and OS/2 share a common 16-bit API. Windows 95 and Windows NT also share a common 32-bit API. (As of this writing, the Windows 95 user interface for Windows NT is in beta test.) 16-bit applications will run under all four versions of Windows. 32-bit applications will NOT run under OS/2 or Windows 3.1.

OS/2 and Windows 95 are the most popular developer platforms. Windows 3.1 and Windows 95 are the most popular end-user platforms. The continuing popularity of OS/2 as a developer platform is a currently a matter of conjecture. Similarly, the absorption rate of Windows 95 by end-users is also in question.

Under Windows 95, 16-bit applications are visually indistinguishable from 32-bit applications. Surprisingly, 16-bit applications are often faster than 32-bit versions because the Windows 95 video support is predominantly 16 bit. Windows 95 (followed soon by Windows NT), however, introduces a new user interface incorporating new controls as defined by a new style guide. Clarion for Windows Version 1.5 implements the Windows 95 style in its own IDE and in both 16-bit and 32-bit target applications. Importantly, both 16 and 32 bit Clarion applications can be created from the same application repository and project file.

With a single development tool that that runs on any platform and produces native applications for any platform, Clarion developers can adopt Windows 95 style immediately while scheduling their platform migration plans (and their end-users) to any comfortable time in the future.

As of this writing, Delphi has no 32 bit support. However, Borland has announced that the Delphi32 will produce only 32-bit targets. Maintaining 16 and 32-bit versions of the same application will require separate projects to be processed with different versions of Delphi.

Debugging

Clarion and Delphi both include full featured Windows hosted debuggers. Both debuggers offer a soft mode where the debugger services repaint events for the target program while it is halted at a break point. The Clarion debugger can be used with both EXEs and DLLs, including DLLs that have been loaded on demand. The Delphi debugger will not debug DLLs.

Internal Performance

Both Clarion and Delphi produce compiled applications with internal performance similar to applications written in C. Clarions optimizing code generator produces faster internal speed but Delphis Visual Component Library produces faster video performance.

Parallel DOS Development

TopSpeed offers both Windows (Clarion for Windows) and DOS (Clarion for DOS) tools for database development. These products contain similar (but not identical) application generators, data dictionaries, and project system. The conversion tools available in both packages permit a developer to maintain both Windows and DOS versions of a Clarion application. There is no DOS version of Delphi.

Conclusions

Clarion for Windows and Delphi differ markedly in style. CW is a state-of-the art *conventional* development environment. The Clarion 4GL has been updated with the latest and greatest abstractions for creating graphic user interfaces, WYSIWYG reports, and database views in a *conventional* language setting. This design leverages legacy software development skills and reduces the learning curve for adapting to graphical client/server technology. Clarions crown jewels, however, are its data dictionary and application generator. These rapid application development tools provide a productivity boost that is available for the entire useful life of a database application.

Delphi, on the other hand, is a state-of-the-art implementation of *object-oriented programming*. OOP is particularly well suited to window controls. Delphi comes with a comprehensive set of Object Pascal window controls available with source code. These controls can be modified and supplemented within the Delphi environment. The resulting versatility in producing imaginative custom user interfaces is Delphis strong point.

In addition to developers style, the choice between Clarion and Delphi should be driven by the type of application to be written. In particular, certain applications favor Clarion for Windows:

Shrink-wrapped Applications

CW is the preferred tool for developing shrink-wrapped applications that contain printed reports or access a database. A Delphi developer must deploy the Borland Database Engine and/or ReportSmith along with his application. These products require separate install processes and 2 to 7 additional diskettes.

Financial Applications

Certain financial calculations cannot be implemented reliably using floating point arithmetic. In particular, rounding currency is a problem. For example, 25.5 cents should be rounded up to 26 cents. The floating point representation for this amount should be .255. However, because exact decimal values sometimes produce irrational floating point values, this value can actually be .25499999... which will be rounded down to 25 cents. Clarion, which uses exact BCD arithmetic, is preferable to Delphi for applications that require absolute accuracy.

Accounting Companion Applications

Many accounting applications use Btrieve to maintain their underlying database. Clarions Btrieve Database Driver accesses these files using the native Btrieve API. In addition, the Btrieve driver will create a Clarion database dictionary from Btrieve record definition (DDF) files. Clarions Btrieve support is far superior to Delphis.

Large Database Applications

A large database application contains hundreds of procedures that may require maintenance for years on end. Clarion visual development tools are designed to manage extremely large application repositories for their entire useful life. Changes to a database dictionary are automatically distributed throughout an application.

This technology is vastly superior to Delphis project management strategy. Delphi experts can only be used once. Program maintenance must then be performed on generated code. Database changes must be made manually--a painstaking task that may be unreasonable for large applications. No abstraction is offered for managing large projects which exist as a long list of cryptic filenames.