Tech Note

embarcadero®

既存の Delphi/C++Builder アプリケーションの移行方針

エンバカデロ・テクノロジーズ

2011 年 11 月

エンバカデロ・テクノロジーズ

〒102-0072 東京都千代田区飯田橋 4-7-1 ロックビレイビル 8F
TEL 03-4577-4520　FAX 03-6843-0961

既存アプリケーションメンテナンスの需要

現在、多くの企業が既存アプリケーション資産を有効活用して開発コストの削減を行おうとしています。あるケースでは、現存のアプリケーションを修正して現在の業務に合うようにしたり、既存のコードを再利用して開発コストを圧縮しようと試みています。この傾向は、単に経済状況に起因するだけでなく、クライアント・サーバーアプリケーションや初期の Web アプリケーションが、ハードウェアやベースとなるソフトウェア（ＯＳやデータベース、アプリケーションサーバーなど）のリプレースの時期を迎えているため、その数が増大しているともいえます。

ダウンサイジングの波に乗って、PC が企業システムのクライアント環境として普及しはじめた頃と違い、アプリケーションを構成するコンピュータは、性能が飛躍的に向上しているとはいえ、大きな変化はないように感じてしまうことがあります。例えば、Windows 95 や Windows NT で構築したクライアント・サーバーアプリケーションを、最新環境に移行したいと考えたときに、おそらく多くのケースでは、ＯＳやデータベースのバージョンが変わる以外、特に変更することはないという印象を持つかもしれません。しかし、実際には、コンピューティング環境は、ここ 10 年で大きく変化しています。

Delphi および C++Builder は、Windows プラットフォームの普及とともに、クライアント・サーバーアプリケーションやデスクトップアプリケーション構築の分野で広く利用されてきました。これらの既存アプリケーション資産の多くは、現在も一部が利用されていたり、何回かのバージョンアップを経て、現役のシステムとして稼動しています。これらのシステム資産を有効に新しいシステムに移行できれば、低コストで新しい環境や業務形態に対応することが可能になります。

本書では、これらの既存 Delphi / C++Builder アプリケーション資産を、新しいシステムに移行していく上で、基本的な方針を決定するためのポイントを解説します。

ゴールの設定

移行プロジェクトにおいて「解決すべき問題は何か？」と質問すると、しばしば「既存のコードが最新バージョンのツールで再コンパイルできること」と真っ先に答えるケースに出会います。このことは、実作業で問題となる部分には違いありませんが、本質的な目標と手段を混同しています。

プロジェクトの最終的なゴールを正しく認識するには、「プロジェクトが成功した結果、どのようなメリットを享受できるのか？」という質問に答えてみることが重要かもしれません。これによって、例えば、単に「Windows 7 で動作するようにすること」なのか、「業務プロセスの変更に対応できるようにすること」なのかといったゴールが見えてきます。

移行プロジェクトでは、既に動作する（あるいは動作していた）既存システム資産があり、スタート地点が白紙ではありません。そのため、どうしても既存システムを円滑に新しい「環境」で動作させることに目を奪われ、その結果得られるメリットや、その次のステップを忘れがちです。

既存アプリケーション資産を円滑に、かつ効果的に移行させるには、まず、ゴールを明確に設定し、その上で、スタート地点から、何を捨て去り、何を加え、何を変更しないかを検討する必要があります。

まず、ゴールを明確にする上で、以下のような各領域について、プロジェクトのゴールを列挙してみましょう。

- **ビジネス上のゴール：**既存のシステムが前提としているビジネスプロセスに対して、何らかの変更を加える必要があるのか？ あるいは、将来的にどのような変更の可能性があるのか？
- **システム環境上のゴール：**新しいシステムは、どのようなハードウェア、およびソフトウェア環境で動作する必要があるのか？ OS、データベースなどのバージョンは何か？
- **性能上のゴール：**新しいシステムは、既存のシステムに対して性能上どのような改善が必要なのか？ パフォーマンス、セキュリティ、可用性など、具体的にどの部分に対してどのような改善を期待されているのか？
- **ユーザーエクスペリエンス上のゴール：**新しいシステムは、既存のシステムに対してユーザーの操作性に関してどのような改善が必要なのか？ 新しいユーザーインターフェイス、新しい入力デバイスへの対応、Web アクセス、モバイルなど、具体的にどのような改善を期待されているのか？
- **コネクティビティ上のゴール：**従来システムと異なり、今日のシステム環境は、さまざまな他のシステムとの接続性が要求されます。どのようなシステムと接続する必要があり、どのようなプロトコルが要求されるのかを理解しておく必要があります。具体的なスペック、システムのどの機能が関連するのか？

現状の把握

ゴールを理解したら、現状の既存アプリケーションとのギャップを検討します。このために、現状の把握が重要になりますが、これらは一般的にゴールに依存します。ゴールを把握する際に検討した各領域における要求が、アプリケーションの具体的などの領域に関係してくるかを、まず理解しましょう。

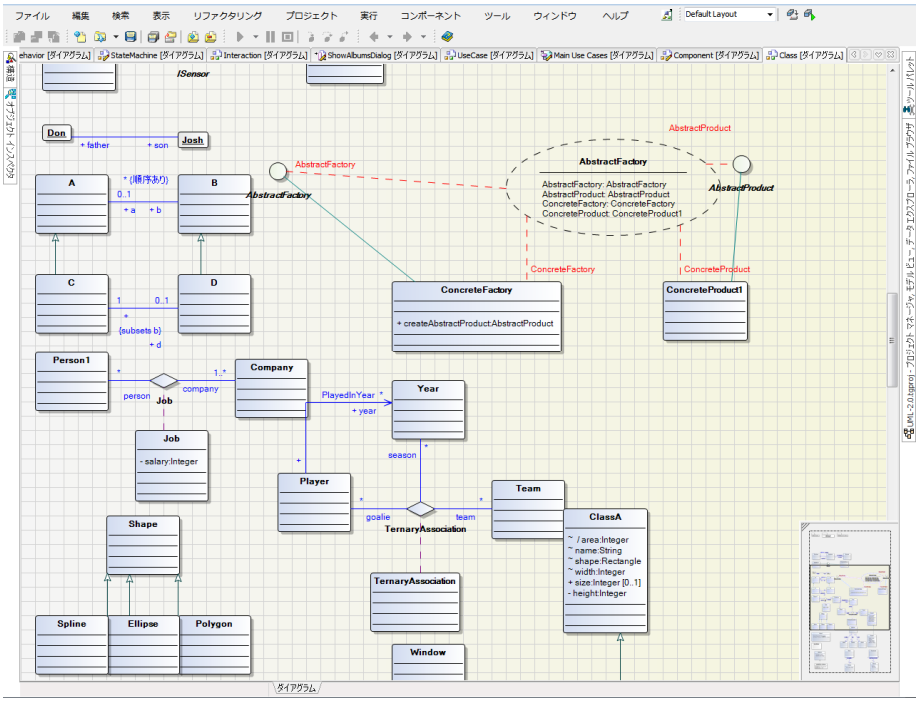
- **ロジックの構造：**ビジネスプロセスに変更があったり、今後、状況の変化に応じて頻繁に要件が変わることが予想される場合、アプリケーションロジックのメンテナンス性は、作業工数に大きく影響します。ロジックの分離性、独立性は、この指標になります。
- **ユーザーインターフェイスのコンポーネント性：**多くのアプリケーションでは、多数のダイアログが存在しているでしょう。特に、従来のステップ型（俗に言う「紙芝居型」）の画面インターフェイスを採用している場合、いわゆる「画面数」はかなりの数になります。これらが、変更にどれだけ対応できるのかは、その画面構成要素やベースとなるウィンドウがコンポーネント化されているかどうか、あるいは継承などによって再利用性を高める配慮がなされているかどうかに関係します。
- **コードの標準化：**コーディングスタイル、クラス設計などといったコードの質に関する問題は、潜在的なバグ、コード改修による予期せぬ動作などに関係します。
- **データベースアクセスアーキテクチャ：**Delphi / C++Builder には、さまざまなデータベースアクセスコンポーネントが存在します。これらのうち、どの技術を使用しているのかを理解することは重要です。同時に、こうしたデータベースアクセスが、どれだけ分離性を有し、再利用やメンテナンスに対応できる構造になっているかが、ポイントになります。
- **データベース設計：**アクセスするデータベースの設計も重要なポイントです。データベースの構造は、アプリケーションの性能やメンテナンス性に大きく関与するため、現状のデータベース設計を理解することは重要です。

既存のアプリケーション資産は、必ずしも現在の要求に対応できるように設計されているわけではありません。これらの現状把握とギャップの検討が、正しいプロジェクトの方向性を導き出します。

次に、具体的な現状把握のためのツール支援について紹介します。

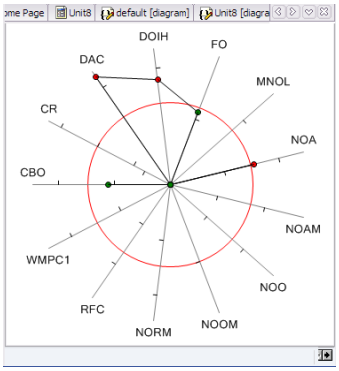
アプリケーション設計の把握

Delphi / C++Builder には、コードからクラス図を作成できる UML モデリング機能が搭載されています。この機能を用いることで、既存アプリケーションのオブジェクト指向設計をビジュアルに把握できます。



コードの質のチェック

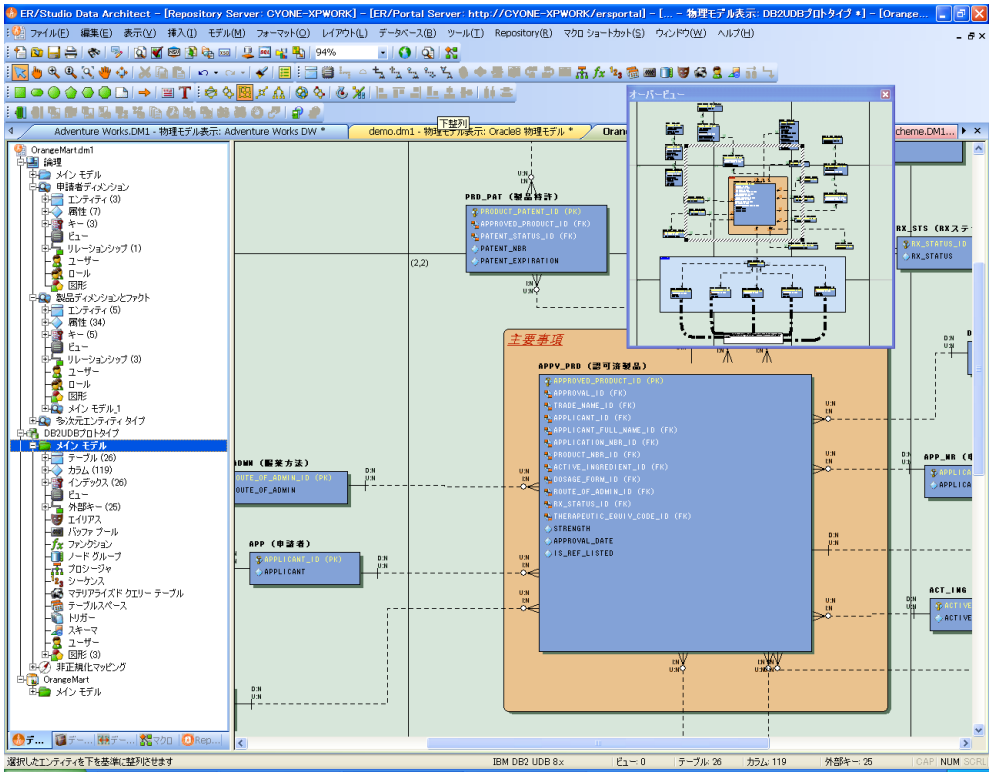
Delphi には、「検査・測定機能」と呼ばれるコードの静的検査機能が搭載されています。この機能を使用すると、既存のアプリケーションコードを定性的、定量的にチェックすることができます。これらのチェックレポートには、潜在的なバグを誘発する要因となるようなコードに対する警告も含まれており、既存のコード全体をレビューする前に、おおよそのコード品質を把握できます。



データベース設計の把握

既存のデータベース設計を理解するには、ビジュアル化が有効です。ER/Studio（Delphi / C++Builder Architect に搭載）を用いると、既存のデータベースからスキーマ情報を読み込んで、データベースの物理設計および論理設計をビジュアル化します。これにより、データベースが現状どのような構造になっており、今後の拡張においてどのような問題が起きうるのかを理解できます。

さらに、図を編集することで、データベースの設計を変更したり、他のデータベース用の物理設計を作成することもできます。変更した設計は、実際のデータベースに反映することができるので、設計の改善や他のデータベースプラットフォームへの移行などにも活用できます。



個別のテクノロジースタックの理解

Delphi / C++Builder アプリケーションで中心的な役割を担うのは、VCL と呼ばれるコンポーネントフレームワークです。これらをどのように利用しているのかを知ることは、現状をより詳細に把握する手助けとなります。

フォームの設計

Delphi / C++Builder アプリケーションを構成するウィンドウは、フォームと呼ばれるオブジェクトです。フォームをどのように設計しているのかによって、これらの再利用性は大きく変わります。

- **画面中心型：**Delphi / C++Builder アプリケーションの開発スタイルは、画面中心といっても過言ではありません。まず、何も配置されていないフォーム上にコントロールを配置し、そのプロパティやイベントを定義することでアプリケーションを開発していきます。その結果、すべてのロジックが画面に「貼り付いている」状態が生まれます。これは、メンテナンス性を考えた場合、あまりよい状態とはいえません。多数の画面を設計しなければならない場合、こうした「画面に貼り付いた」同じようなロジックを何度も記述することになります。これらを整理せずにそれぞれのフォームに記述した場合、類似のコードがアプリケーションの各所に散らばっている状態にあります。
- **フォーム継承型：**画面まわりについてコードの再利用性を整理したのが、フォームの継承を用いたアプリケーションです。Delphi / C++Builder では、自身で作成したフォームを基底クラスとして継承し、新しいフォームを作成することができます。この機能を利用して作成されたアプリケーションは、ユーザーインターフェイスに関するコードやコンポーネントの設定について、再利用性が考慮されているといえます。

- **データモジュール型：**データベースアクセスなどのロジック部分を画面から切り離し、独立したモジュールに配置します。この形態のアプリケーションでは、データアクセスレイヤーを分離しやすいので、異なるデータアクセスアーキテクチャに移行した場合でも、ユーザーインターフェイスの再利用性が高くなります。
- **クラス活用型：**データモジュール以外にも、ロジック部分を独立したクラスとして設計し、画面中心型から脱却しているアプリケーションは、ユーザーインターフェイスからの独立性も高く、高い再利用性があるといえます。同時に、ロジック部分とユーザーインターフェイスをつなぐメソッドもはっきりしているので、ロジック側の変更も容易であると予想できます。

データベースアクセスアーキテクチャ

Delphi / C++Builder のデータベースアクセスアーキテクチャには、多数の選択肢があります。これらの違いを理解することで、現状の課題と移行の方針を検討することが可能になります。

- **BDE：**Borland Database Engine、古くは IDAPI と呼ばれたデータベースアクセスエンジンです。BDE を使用したデータアクセスコンポーネントには、TTable、TQuery などがあります。このアーキテクチャは、元来デスクトップデータベースアクセス用に開発されたもので、Paradox や dBase などのアクセスに利用してきました。Delphi の初期のバージョンで、RDBMS に接続するための技術としても拡張されましたが、リモートデータベースをあたかも手元のデスクトップデータベースのように扱うその手法は、簡単である反面、システムには大きな負荷をかける遠因になることがあります。
- **dbExpress：**BDE が、クライアント環境にエンジンソフトウェアをインストールしなければならないのに対し、dbExpress はより軽量なドライバのみで動作します。クライアント環境のメンテナンス負荷を軽減するとともに、システムに対して余計な負荷をかけないシンプルな設計となっています。一方、従来アプリケーションでは、BDE が提供してきたさまざまな「デスクトップデータベースをエミュレーションする機能」を利用しているケースがあり、これらをそのまま dbExpress に持ち込むことは、むしろ性能上の問題を引き起こしかねません。
- **InterBase Express (IBX)：**InterBase Express は、InterBase データベースのデータにアクセスするためのデータアクセスコンポーネントのセットです。InterBase Express の各コンポーネントは、InterBase Client API と直接に相互通信し、データ処理を行います。そのため、配布の際、追加のドライバは必要ありません。
- **dbGo (ADOExpress)：**dbGo は、OLE DB プロバイダを介してデータにアクセスする COM オブジェクトをラップしたコンポーネントです。実際のデータベースへのアクセスは、データベースプロバイダ等から供給されている OLE DB プロバイダまたは ODBC ドライバ、使用される特定のデータベースシステムのクライアントソフトウェアによって行われます。そのため、移行にあたっては、これらのドライバソフトウェアとの関係を確認する必要があります。

BDE は、元来デスクトップデータベースを前提に作られたアーキテクチャなので、その手法をそのまま SQL データベースアプリケーションに適用している場合には注意が必要です。SQL データベースを用いる場合の基本的な方針は、サーバー側で適切な処理を行い、クライアント側には最小限の結果セットを持ってくるような設計にするということです。例えば、

- **Table コンポーネント：**Table コンポーネントは、実質的に ” SELECT * FROM TABLE” にすぎません。データ量が増加したときに、不要なデータをクライアント側に持ってくる結果になります。SQL データベースでは、必ず Query タイプのコンポーネントを使って、WHERE 節によってデータベースのクエリー機能を使ってデータを取得するようにします。

- **Filter プロパティ：**Filter プロパティは、データベースサーバーから取得した結果セットに対して、フィルター操作をかけます。もし、フィルターをかけた結果のデータしか使わないのであれば、サーバー側でフィルタリングしておくべきです。つまり、WHERE 節を指定するということです。
- **RecordCount プロパティ：**RecordCount プロパティも、結果セットに対してレコード数をカウントします。データベースのレコード数を得るためにすべてのデータをサーバーから取得するのはナンセンスです。”SELECT COUNT(*)” で十分です。もし、結果セットを for ループなどでフェッチする場合には、RecordCount を参照してカウンター変数をチェックするのではなく、 Eof プロパティで次のレコードがあるかどうかを確認できます。
- **SQL を何度も発行して結果が得られるような処理：**何回も SQL を発行してやっと結果が得られるような処理については、ストアドプロシージャやビューを使って実装できないか考えましょう。最終的な結果が得られる前に、何度もクライアント／サーバー間でデータが行き来するのは効率的ではありません。

データベースアプリケーションで、もうひとつ注意すべきなのは、トランザクションの扱いです。特に従来 BDE などのデスクトップデータベースアーキテクチャを使用していて、SQL ベースのアーキテクチャに移行する場合には、パフォーマンス低下という問題を引き起こすことがあります。

- **暗黙のトランザクションによるデータベースパフォーマンスの低下：**トランザクションに関する設定を何も行わずに、データベースの 1 レコードの処理を行うと、都度、暗黙のトランザクションが発生します。少量のレコードの処理では、それほど問題はありませんが、大量のデータの追加・更新など行った場合、 1 つのレコードごとに、この暗黙のトランザクションが発生し、パフォーマンスの低下の原因となることが多々あります。大量のレコードの更新を行う場合は、更新処理の範囲を指定した明示的なトランザクションを行うことで、パフォーマンスの低下を防ぐことができます。
- **頻繁なデータベースの接続/解除によるアプリケーションパフォーマンスの低下：**データベースコンポーネントから、データベースに接続した時点で、その後の処理に対する必要な情報をメモリ上に展開する処理が行われます。従って、データベースの接続/解除を頻繁に行うと、アプリケーションのパフォーマンスの低下につながります。 1 つの接続の中で行う処理を効率的に配置することで、処理スピードの低下を防ぐことができます。

デスクトップデータベースから SQL データベースへの移行では、最終的にテーブルの設計を SQL データベースのアーキテクチャに最適化することが求められます。こうした移行作業においては、データベース設計をビジュアルに理解できる ER/Studio などが有効です。

その他の注意すべき課題

Delphi / C++Builder アプリケーションでは、このほかにもいくつかの注意すべき領域があります。これらの事項が、対象とする既存アプリケーションで該当するかどうかも確認してください。

- **Windows 64-bit 版のサポート：** Delphi XE2 では、新たに Windows 64-bit をサポートしています。Windows 64-bit 版の Delphi では、コンパイラ、VCL、RTL のすべてが 64-bit に対応しており、多くのアプリケーション資産は、そのまま 64-bit 環境に移行することができます。アプリケーションの 64-bit 対応によって、64-bit プロセッサの使用、大きなメモリ空間の使用といったメリットが得られます。一方、アプリケーションサイズやポインタのサイズなどが 32-bit アプリケーションと比較して大きくなります。現在の Windows 64-bit 環境では、すべてのアプリケーションについて 64-bit 版を用意しなければならないわけではないことに注意してください。アプリケーションが大幅に 64-bit のメリット

を活用できるのであれば、積極的に 64-bit 対応を進めるべきですが、OS が 64-bit 版であるからという理由だけで、64-bit 化を行う必然性はありません。
Delphi XE2 では、比較的スムースに 64-bit プラットフォームに移行できるように設計されていますが、実際の移行作業では特に以下の点について注意が必要です。

- ポインタの操作、演算などで、ポインタのサイズを 4 バイトと想定していないか（64-bit プラットフォームでは 8 バイト）？
- インラインアセンブリコード（64-bit プラットフォームではアセンブリコードと Pascal コードを 1 ブロックに混在できない）
- Windows API の呼び出し
- レコード型のデータフィールドのアラインメント

64-bit プラットフォーム向けの開発における注意点の詳細については、製品ドキュメントの「Windows 向けの 64 ビットクロスプラットフォームアプリケーション開発」の項目をご覧ください。

- **AnsiString から UnicodeString への移行：**Delphi / C++Builder 2009 以降では、標準の文字列型が AnsiString から UnicodeString へ変更されました。Windows では現在、Unicode を標準文字コードとして採用しているため、これは OS プラットフォームに合致した変更です。ライブラリでは、Unicode 化に伴う書き換えを最小化するように互換性を配慮した設計を行っていますが、いくつかの箇所で注意すべき点があります。これについては、Unicode への移行のためのホワイトペーパーを確認してください。
- **サードパーティコンポーネント：**サードパーティ製のコンポーネントを利用している場合、これらをどのように移行させるかは、一般的な Delphi / C++Builder の移行に関するテクノロジースタックとは別に検討しなければなりません。移行ターゲットとなる Delphi / C++Builder に対応したコンポーネントがリリースされているかどうかも重要ですが、これらのコンポーネントが、移行先のシステムでどの程度重要なのかも見極める必要があります。代替となる標準コンポーネントが存在していたり、現在のユーザーインターフェイスのトレンドでは、必要としないコンポーネントかもしれません。
- **レポートツール：**レポートツールは、アプリケーションのアウトプットを担う重要な機能を担当しています。現状のレポート資産をそのまま移行できるに越したことはありませんが、データベースアクセスなどの他のテクノロジースタックとの関係にも注意しなければなりません。また、従来型の「帳票出力」に対し、現在要求されている「レポート機能」とのギャップも整理しておく必要があるかもしれません。一般的に、日本における「帳票出力」に対する機能要求は、海外のものよりも高く、そのため非常に詳細なカスタマイズに対応した帳票ツールが日本市場には普及しています。従来の Delphi / C++Builder にバンドルされていたレポートツールである Quick Report は、現在 QBS Software 社から提供されており、最新バージョンに対応した製品もリリースされています。日本では ComponentSource（http://www.componentsource.co.jp/）より購入できます。
Delphi / C++Builder / RAD Studio XE2 には、新たに FastReport が搭載されています。FastReport は、コンポーネントベースで高品質なレポートを作成できるツールです。日本語版の発売も予定されており、次世代レポートツールとして検討に値するでしょう。
- **マルチプラットフォームサポート：**Delphi / C++Builder XE2 に新たに搭載された FireMonkey フレームワークを用いれば、Windows/Mac OS X の双方で動作するアプリケーションを構築できます。複数の OS 環境をサポートする必要がある場合には、既存アプリケーション資産を FireMonkey フレームワークに移行することを検討する必要があります。FireMonkey フレームワークは、VCL とは異なるフレームワークレイヤーを使用しているため、多くのユーザーインターフェイスは、新しいフレームワーク用に再構築する必要があります。

- **モバイルデバイスのサポート：**スマートフォンなどのモバイルデバイスからシステムにアクセスしたいという要求は、もはやビジネスアプリケーションの世界でも一般化しつつあります。RAD Studio XE2 が提供するテクノロジーを用いれば、さまざまな方法でモバイルデバイスをサポートできます。RAD Studio XE2 では、以下の方法でモバイルデバイスをサポートします。

  - Delphi / C++Builder XE2 の DataSnap：
    DataSnap モバイルコネクタを用いれば、Delphi / C++Builder で開発したサーバーアプリケーションに、iOS、Android、BlackBelly、Windows Phone といったモバイルデバイスからアクセスできます。DataSnap モバイルコネクタは、これらのデバイスからアクセスするためのアダプタコードを生成します。開発者は、このアダプタコードを使って、それぞれの開発言語（Objective C、JavaScript、C#など）から、容易に Delphi / C++Builder によって実装されたロジックを呼び出すことができます。

  - Delphi の iOS サポート：
    Delphi XE2 と FireMonkey の組み合わせによって、iOS 向けアプリケーションを開発することができます。Windows の Delphi IDE でアプリケーションを構築し、これを Mac 環境の X-code でコンパイルすれば、iOS で動作するネイティブアプリケーションを作成することができます。

  - RadPHP によるモバイルアプリ開発：
    RadPHP XE2（RAD Studio XE2 に含まれます）を用いれば、モバイルデバイスに最適化された Web アプリケーションを簡単なビジュアル操作で開発できます。また、AppStore で配布可能なスタンドアロンアプリを構築することも可能です。RadPHP は、Delphi と同じようなコンポーネントによるビジュアル操作による開発をサポートしているので、Delphi / C++Builder ユーザーも、比較的簡単に開発をスタートできます。

アプリケーション形態の変化に対する理解

冒頭に述べたように、コンピューティング環境は、ここ 10 年で大きく変化しています。特に、10 年前のシステムは、LAN 環境で独立したアプリケーションとして動作する形態が多く、実質的に「他とつながらない」システムでした。しかし、現在では、さまざまな連携が要求されます。ERP との連携、CRM からのデータインポート、あるいは外部の Web サービスの利用など、異なるアーキテクチャで作られたシステムとつながるのが現在のシステムです。

これらのトレンドは、対象となるアプリケーションの移行に影響を与えるでしょうか？ 答えは、はじめに理解したゴールに関係します。

例えば、対象となるアプリケーションが他のシステムとの連携を必要とする場合、そのシステムとのインターフェイスをどのように実装するかは、移行作業に大きな影響を与えます。低レベルなレイヤーでは、通信プロトコル、文字コードなど、より高度なレベルでは、セキュリティ、トランザクション、同期などの問題を考慮しなければなりません。また、これらがパフォーマンスに与える影響なども検討しておく必要があります。

システムの利用形態が拡大するからといって、従来のアプリケーション資産をすべて書き直す必要が生じるわけではないことに注意してください。現在のシステムは、複数のサブシステムが並列して大きなシステム系を作るアーキテクチャです。このシステム系にうまく従来システムをあてこめば、いくつかの従来資産をそのまま活かしながら、新しい追加機能を段階的に導入していくこともできます。

移行プロジェクトにおけるツール環境

エンバカデロでは、移行プロジェクトにおいて生産性を上げることができるコストパフォーマンスの高いソリューションを用意しています。

以下は、移行プロジェクトの各フェーズで、メインの開発環境とは別に有効なツールです。

- **ER/Studio Data Architect：**ER/Studio は、データベース設計をビジュアル化し、適切な構造への改善を支援するデータモデリングツールです。既存アプリケーションのデータベース設計を最適化し、新しいプラットフォームに移行する際に役立ちます。不適切なデータベース設計は、アプリケーションの拡張性、メンテナンス性、パフォーマンスに大きく影響します。移行とともに、よりよいシステムに生まれ変わらせるには、ER/Studio の活用が必須です。
- **DB Optimizer：**アプリケーションパフォーマンスにおけるデータベースアクセスが占める割合は、大変大きいものがあります。特に、膨大なデータを扱うケースでは、その差は n 倍に膨れ上がります。DB Optimizer は、データベースパフォーマンスを開発段階から分析、特定するツールです。これにより、特定の SQL 文がパフォーマンス上の問題を抱えており、どの部分にもっとも時間を消費しているかを、ビジュアルに掌握することができます。改善のための代案も提示され、実際にその効果を測定できるので、効果的なパフォーマンス改善が可能です。

また、これらのツールや、移行に伴い必要となる複数バージョンへのアクセスを提供するライセンスオプションも用意しています。

EMBARCADERO® ALL-ACCESS™

Embarcadero All-Access は、Delphi、C++Builder をはじめとするエンバカデロの開発ツール、ER/Studio や DB Optimizer などのデータベースツールすべてにアクセスできるツールセットです。All-Access は、およそ 2、3 種類のツールを購入するぐらいのコストで、エンバカデロのすべてのツールと、複数のバージョンの利用が可能なので、移行プロジェクトなどで、ER/Studio などを使用したい場合、また、複数バージョンの Delphi や C++Builder を必要とする場合に便利です。All-Access には、複数ユーザーでライセンスをシェアすることのできるネットワークコンカレントライセンスも用意されているので、これらのライセンスと通常の開発ツールライセンスを組み合わせることで、ツール環境を準備するコストを削減することができます。

開発ツールのネットワーク型ライセンスオプション

開発プロジェクトでは、しばしば人員の入れ替えや増減が発生します。このようなときに、従来型の指名ユーザーライセンスでは、使用者の変更や管理が煩雑になってしまいます。エンバカデロでは、AppWave と呼ばれるソフトウェア配信・管理のテクノロジーを提供しており、主要なツールについて、ネットワーク型のライセンスオプションを用意しています。同時使用数を規定した複数ユーザーによるツールの共有、複数バージョンの使い分け、時期によって使用者を変更するための設定などを、集中管理されたサーバーによって行うことができます。

アプリケーションの配布と管理

開発したアプリケーションをユーザー環境に配布し、適切に管理していくことは、円滑にシステムを運用していく上で重要です。しかし、多くの場合、クライアント環境を完全にコントロールできないことから、アプリケーションの配布や管理により多くの時間を費やしてしまいます。

Web アプリケーションやクラウド型のアプリケーションは、こうした苦労を軽減するためのひとつのアプローチですが、Delphi や C++Builder のような、高機能で操作性の高いデスクトップアプリケーションにこれらの技術を適用すると、性能を犠牲にせざるを得ません。

エンバカデロでは、こうしたアプリケーションの配布と管理を軽減するためのソリューションとして、AppWave を提供しています。AppWave は、エンバカデロツールだけでなく、任意のアプリケーションを配信、管理できる技術です。開発したアプリケーションを「AppWave アプリ」にパッケージ化すれば、インストール操作なくアプリケーションを利用できる「ストリーミング配信」が可能になります。ユーザーは、一切のインストール操作をすることなく、ワンクリックで、アプリケーションを素早く起動できます。

「AppWave アプリ」に変換する作業は、AppWave Studio という専用のツールを使います。ウィザード形式の簡単な操作によって、アプリケーションのインストール作業をキャプチャし、仮想的にアプリケーションをインストールした状態をつくり、ワンクリックで実行可能にします。特別なコードを記述したり、アプリケーションを修正する必要はありません。

AppWave は、既存のアプリケーションにそのまま適用して、インストールに関連するさまざまな問題（バージョン間の衝突、冗長なセットアップ作業など）を解消することも可能です。

AppWave の詳細については、http://www.embarcadero.com/jp/appwave をご覧ください。

エンバカデロ・テクノロジーズについて

エンバカデロ・テクノロジーズは、1993 年にデータベースツールベンダーとして設立され、2008 年にボーランドの開発ツール部門「CodeGear」との合併によって、アプリケーション開発者とデータベース技術者が多様な環境でソフトウェアアプリケーションを設計、構築、実行するためのツールを提供する最大規模の独立系ツールベンダーとなりました。米国企業の総収入ランキング「フォーチュン 100」のうち 90 以上の企業と、世界で 300 万以上のコミュニティが、エンバカデロの Delphi®、C++Builder®、JBuilder®といった CodeGear™製品や ER/Studio®、DBArtisan®、RapidSQL®をはじめとする DatabaseGear™製品を採用し、生産性の向上と革新的なソフトウェア開発を実現しています。エンバカデロ・テクノロジーズは、サンフランシスコに本社を置き、世界各国に支社を展開しています。詳細は、www.embarcadero.com/jp をご覧ください。