Firebird White Paper

Warum typische Delphi Projekte oft scheitern ...

Eine subjektive Betrachtung von Holger Klemt, Oktober 2014

Viele von Euch kennen mich schon seit Jahren und einige haben mich kennengelernt als Delphi Entwickler, der in Vorträgen auf diversen Konferenzen diese Plattform immer wieder gerne vorgestellt hat.

Angefangen habe ich mit einer frühen Delphi 0.9 Beta Version im Jahr 1995 und war begeistert. Zu dieser Zeit war das Internet noch nicht überall präsent, Windows 95, Compuserve und AOL brachten aber schon einiges in Bewegung. Damals waren die Messen wie CeBIT in Hannover und Systems in München noch die hauptsächlichen Veranstaltungen, um sich mit wichtigen Informationen zu versorgen.

Auf der Suche nach brauchbaren Alternativen für die bis dahin von mir für Kunden erstellten Datenbank-anwendungen auf Basis von Turbo Pascal, dBase und später dann Microsoft Access, bot mir Delphi von Anfang an alles, was ich für meine Projekte brauchte. Turbo Pascal beherrschte ich schon länger, daher war Delphi meine erste Wahl.

In Zusammenarbeit mit dem damals sehr aktiven Borland Distributor better office ergaben sich interessante neue Chancen. In den Jahren ab 1996 war ich dann auch Teil des Borland CeBIT Teams und habe vor unglaublichen Zuschauermengen die jeweils neuen Delphi Versionen im Softwarekino vorgestellt. Weitere Vorträge auf diversen Konferenzen in der ganzen Welt machten wirklich Spaß und die positive Resonanz der Zuschauer war enorm.

Es reichten wirklich die wenigen Minuten, um die Zuschauer von der Leistungsfähigkeit und vergleichsweise unglaublichen Produktivität zu überzeugen. Neues Projekt erzeugen, TDatabase hinzufügen, verbinden mit DBDemos; TTable hinzufügen, verbinden mit einer Tabelle; TDatasource dazu, TDBNavigator dazu, TDBGrid aufnehmen, mit dem Feldeditor der TTable die TField Objekte erzeugen und diese per Drag and Drop ins Formular ziehen, kompilieren, starten, Wow. Die Zuschauer standen oft mit offenem Mund und großen Augen vor uns und konnten kaum erwarten, die jeweils neueste Delphi Version zu erwerben, egal, was es kostet.

Die Verkaufszahlen waren damals sicherlich beeindruckend, damit hatte ich aber gar nichts zu tun. Ich war sozusagen Showprogrammierer und kein Verkäufer. Aus meiner Perspektive war diese Entwicklungs-umgebung damals das Nonplusultra und ich habe gerne in diversen Schulungen und Vorträgen weltweit die vorhandenen Techniken vorgeführt und aufgezeigt. In dieser Zeit sind viele Kontakte entstanden, von denen mir auch heute eine große Anzahl erhalten ist. Durch Beratungs- und Programmieraufträge konnte ich diverse Kundenprojekte aktiv mitgestalten. Ein großer Teil dieser Projekte sind auch heute noch im Einsatz.

Zu diesem Zeitpunkt begannen aber bereits Softwarehäuser, Entwicklungsabteilungen und Einzel-entwickler, sich mit genau diesem komponenten- und formularbasierenden System in eine Sackgasse zu begeben, die durch eine flexible Architektur hätte vermieden werden können.

Das war das goldene Zeitalter der Komponentenhersteller und so manches Softwarehaus hat dabei alle Komponenten eingesetzt, die damals gerade so angesagt waren.

Neue Delphi Versionen setzten auch immer neue Versionen der Komponenten voraus, so dass viele Updates erst dann möglich waren, wenn die Komponentenhersteller auch ein Update lieferten.

Solange die Komponentenhersteller im Delphi Umfeld ausreichend Geld verdient haben, war das auch kein Problem. Viele Komponentenhersteller waren auch in den Beta Test Newsgroups aktiv, so dass die Komponenten sicher auch schon beim Erscheinungstermin oder kurz danach verfügbar sein sollten.

Nachdem ich mich in den ersten beiden Jahren auch in dieser Sackgasse befand, ging ich immer mehr dazu über, meine Projekte mit einer generellen Architektur umzusetzen, die ich bereits auf einigen Entwickler Konferenzen unter dem Titel OOP vorgestellt habe.

Diesen Ansatz habe ich dann als programmierender Softwarearchitekt in verschiedenen, langjährigen Kundenprojekten umgesetzt. Von unseren Kunden wird dieses als Basistechnologie nach wie vor eingesetzt und teilweise selbstständig und aktiv weiterentwickelt.

Das oben erwähnte Beispiel, das wir auf der CeBIT und auf der Systems gezeigt haben, verdeutlicht eigentlich schon einen großen Teil der Problematik. Wir haben das damals aus Überzeugung vorgeführt, weil es unglaublich schnell zu einem benutzbaren Prototyp führte.

Abhängigkeiten

Die Datenbankstruktur wird auf Datenbankebene mit einem geeigneten Datenbank Editor festgelegt, d.h. die Kundentabelle wird zum Beispiel mit einem Tabellenfeld für die Postleitzahl definiert, hinter der dann als Fremdschlüssel eine Postleitzahlentabelle definiert wird. Die Postleitzahl ist dabei oft ein Zwangsfeld. Einige sind seit der Wiedervereinigung ein wenig sensibler geworden, da vierstellige Postleitzahlen auf fünfstellige umgestellt wurden neben der Aufgabe, führende Nullen zuzulassen. Also wurde kein Integer-Datentyp, sondern ein Textfeld festgelegt. In Delphi erstellt man nun für genau diese Struktur ein Formular oder ein Frame, gerne auch gleich mit TDataset, TDatasource, TDBEdit und TDBLookup-Combobox drauf und hängt noch diverse Events hinter die Datasets und verdrahtet am Ende die Feldtypen auch noch direkt im Feldeditor.

Vorteilhaft ist an dieser Vorgehensweise, dass sehr schnell ein Ergebnis erzielt wird, nämlich eine für den Endanwender benutzbare Anwendung mit echten Formularen und echter Datenbankanbindung. Leider existieren aber nicht nur Kundenadressen, sondern auch Lieferantenadressen, Mitarbeiteradressen etc. Das hat zur Folge, dass für jedes Formular die passende Tabelle wieder mit den gleichen Hilfsmitteln erzeugt werden muss.

Ereignisse mit gleichen Aufgaben verteilen sich überall im Quellcode in diversen Units. Techniken, wie Formularvererbung versprechen dabei Vereinfachungen, die aber auch leider am Ende das Gegenteil bewirken können. Die Auslagerung der TDatasets auf Datenmodule entlasten die Elemente im Formular, bringen aber in vielen Fällen gerade im Team neue Probleme.

Solche Projekte scheitern dann schon gerne mal an ganz einfachen Anforderungen im Bereich Globalisierung. Wer kann denn schon ahnen, dass es zum Beispiel in Irland keine Postleitzahlen gibt, also ein Zwangsfeld dafür durch einen Dummy ersetzt werden muss.

Das gesamte Verfahren, das auf den bisherigen Kenntnissen basiert, muss nun in allen Formularen angepasst werden. Gleichzeitig muss auch noch die Struktur in der Datenbank an die neuen Anforderungen angepasst werden. Wer im gleichen Schritt die Postleitzahlenfelder vom **String[5]** auf zum Beispiel **String[10]** und "nicht Zwangsfeld" umstellt, deckt laut Wikipedia zwar die weltweiten Anforderungen an Postleitzahlen ab, wird aber schnell feststellen, dass eventuelle Einträge im TDataset Feldeditor weiterhin die alte Länge erwarten und sich gar nicht erst ohne Fehlermeldung öffnen lassen. Ebenso müssen die Spaltenbreiten in TDBGrids in allen Formularen an die neuen Anforderungen angepasst werden.

Das Programmierteam ist tagelang mit Anpassungen und Tests der neuen Anforderungen beschäftigt. Bei Softwareherstellern muss auch noch ein geeignetes Verfahren für einen Rollout ohne Datenverlust erarbeitet werden.

Das Problem an vielen so aufgebauten Delphi Anwendungen ist die direkte statische Abhängigkeit der Benutzeroberfläche von den Datenbankstrukturen. Selbst so banale Änderungen, wie die Verlängerung eines Feldes an den Datenstrukturen, werden nicht vom Delphi Programm übernommen. Diese sind in diversen Referenzen in vielen Formularen verteilt und müssen dort vom Entwickler an diesen Stand angepasst werden.

Softwarearchitektur Teil 1

Im Rahmen meiner EKON OOP Sessions um die Jahrtausendwende, an die sich, im Hotel in Mörfelden in hoffnungslos überfüllten Seminarräumen teilweise sogar auf dem Fußboden sitzend, sicher noch manche(r) erinnern kann, zeigte ich dabei einen einfachen Mechanismus auf, mit dem sich große Teile dieser Problematik bereits vermeiden ließen.

Der zentrale Dreh- und Angelpunkt dabei war ein kleines Delphi Programm, welches große Teile meiner Quelltexte auf Basis der Tabellenstrukturen erzeugt hat. Zu jeder Tabelle gab es eine eigene, automatisch erstellte Unit, die alle Tabellenfelder in einer Klasse abgebildet hat. Eine immer wieder erweiterte Basisklasse, von der alle automatischen Klassen abgeleitet wurden, sorgte dabei für die Implementation von lesen, schreiben, usw. Nur dort befanden sich Referenzen der benutzten Datenbankkomponenten. Diese konnten dann nicht nur einfach ausgetauscht werden, sondern sogar mit einfachen Hilfsmitteln auf mehr oder weniger exotischen Plattformen wie die IBM AS/400 iSeries arbeiten, ohne dass sich Entwickler um etwaige Besonderheiten kümmern mussten.

Nahezu alle SQLs wurden dabei automatisch generiert. Weiterhin hat der Quelltextgenerator dann noch für jede Tabelle eine weitere, davon abgeleitete Extended Class in einer eigenen Unit erzeugt, die auch nur einmalig erstellt wurde. In diesen Units konnte der Entwickler dann Methoden überschreiben, da diese nicht vom Quelltextgenerator überschrieben wurden. Nach jeder Datenbankänderung wurde der Quelltextgenerator wieder angeworfen und der Delphi Compiler hat dann schnell aufgezeigt, ob irgendetwas nicht mehr kompatibel ist.

Standardaufgaben, wie Datensätze lesen, schreiben, löschen, anzeigen oder auswählen, wurden durch die Basisklasse implementiert. Nur dort waren dann auch individuelle TxxGrids oder ähnliches referenziert, so dass spätere Änderungen jederzeit möglich waren. Komplexe Formulare überschrieben dabei in der Extended Class alle relevanten Methoden und ersetzten diese bei Bedarf durch Standardformulare. Daher konnten auch ganz klassisch erzeugte Formulare Teilaufgaben übernehmen, aber in ihrer Implementation auch wieder auf die abgeleiteten Klassen zugreifen.

Das klingt komplizierter als es ist. Der Quelltextgenerator ist im Laufe der Jahre mit der Basisklasse gewachsen und wurde von den Kunden meistens selbst erweitert. Bei einem Kunden, bei dem der Quelltextgenerator für über 800 Tabellen ca. 1600 automatisch erstellte fehlerfreie Interface Units auf Knopfdruck erzeugt hat, waren die Vorteile dieses Verfahrens schnell klar.

Die Lösung des o.a. Problems war in dieser Architektur daher sehr schnell erledigt, die Tabellen wurden in der Datenbank geändert, der Quelltextgenerator wieder angeworfen und schon waren die Interface Units wieder aktuell und alle global definierten Regeln passten zum Datenmodell. Der neu kompilierte Quelltext beherrschte also sofort nach der Datenbankänderung die darin enthaltenen Grundlagen.

Der Nachteil dieser Lösung war die anfänglich relativ lange Entwicklungszeit, die man für die Basisklasse und den Quelltextgenerator brauchte, um alle eigenen Anforderungen umzusetzen. Nachdem das System erstellt war, hat sich die Entwicklungszeit relativiert.

Die Vorteile, dann später dynamisch auf jegliche Datenbankänderung reagieren zu können, führten auch dazu, dass Teile des Entwicklerteams nur in Zusammenarbeit mit der Fachabteilung das Datenbankmodell weiterentwickelt haben, ohne selbst auch nur eine Zeile Delphi programmieren zu müssen.

Mit einer Batchdatei waren der Quelltextgenerator und der Delphi Kommandozeilencompiler schnell angeworfen und so konnte auf Knopfdruck der Fachabteilung das in das Gesamtsystem integrierte neue oder angepasste Formular zum Test übergeben werden. Daraus ergaben sich dann oftmals weitere Änderungswünsche durch neue Erkenntnisse der Fachabteilung, die nun wieder in das Datenbankmodell integriert wurden und zum nächsten Prototyp führten oder neue Anforderungen an das Entwicklerteam, die in der Extended Class umgesetzt werden konnten.

Einsatz in der Praxis

Die damals in einem langjährigen Kundenprojekt entwickelte und verfeinerte Quelltextgeneratortechnik wird bei diesem und anderen Kunden nach meinem Kenntnisstand noch heute eingesetzt und stets weiter verfeinert. Eine Portierung der kompletten Software auf eine neue Delphi Version war schnell gemacht, weil die meisten benutzten Komponenten und Methoden eben durch Anpassungen im Quelltextgenerator und in der Basisklasse schnell erledigt waren.

Natürlich beschränkt dieses Verfahren im gewissen Rahmen die Flexibilität innerhalb der jeweiligen Formulare, aber man verhindert damit den größten Fluch aller Delphi Programme: Alles ist irgendwie voneinander abhängig. Wenn die Datenbank nicht zum Formular passt, gibt es Probleme; wenn die Formulare untereinander nicht passen, gibt es Probleme etc.

Grundlegende Änderungen sind in typischen Delphi Projekten nur unter hohem zeitlichem Aufwand realisierbar und verursachen oft mit der neuen Funktionalität neue Probleme in bereits existierenden und bewährten Modulen.

Abschreckende Beispiele

Wenn nun noch Kunden von Softwarehäusern individuelle Anpassungen anfordern, wird oft zu den Packages als Lösung gegriffen. Von diesen Lösungen halte ich persönlich nicht viel. Das ist sicher mein subjektiver Eindruck, aber auch nach langjähriger Erfahrung komme ich zum selben Ergebnis. Der Rekordhalter, den ich im Rahmen meiner Beratungstätigkeit kennengelernt habe, hatte eine CRM Software

mit ca. 2000 Installationen, in der durch Compilerschalter 80 verschiedene Kundenversionen erzeugt wurden. Das komplette System war völlig unwartbar und die Kunden beschwerten sich ständig über die fehlerhafte Software, über wiederkehrende Fehler, die in neueren Versionen schon behoben waren und fehlende individuelle Anpassungen, die bei neuen Versionen übersehen wurden.

Der Inhaber des Unternehmens, seines Zeichens selbst kein Programmierer, aber ein sehr guter Verkäufer, hatte sich dabei auf das Urteil seines Chefarchitekten verlassen, der ihn lange Jahre davon überzeugt hat, dass es keine bessere Lösung geben würde.

Im Rahmen eines Code Review Tages beim Kunden vor Ort konnten wir schnell feststellen, dass sich hier kein generelles Problem mit dem verwendeten Werkzeug, sondern eher ein selbstgemachtes Problem ergab.

Wie auch hier, habe ich im Laufe meiner Beratertätigkeit gerade im Delphi Umfeld, sehr viele, teilweise hochqualifizierte Fachleute, mit sehr guten Prozesskenntnissen der jeweiligen Branche kennengelernt, die sich dann das Programmieren mit diversen Büchern selbst beigebracht haben. Daraus sind teilweise sehr gute Branchenlösungen hervorgegangen. Die Einfachheit der Pascalsprache sorgte für eine schnelle Einarbeitung. Des Öfteren hörte ich auch "Damit habe ich damals in der Schule/im Studium ein paar Programmiergrundlagen gelernt, daher hab ich dann damit einfach weiter gemacht".

So sind durchaus erfolgreiche Unternehmen entstanden, die mit sehr guten Abbildungen der Prozesse und einer dafür geeigneten und funktionierenden Branchensoftware für viele Kunden der ideale Partner waren. Viele davon stehen aber 10 oder 15 Jahre später vor großen Herausforderungen, um weiterhin im Markt erfolgreich zu bleiben.

Customizing

Durch einzelne Großkunden, die mit einem entsprechenden Budget an die Softwarehersteller herantreten, um sich besondere Ansprüche und Wünsche realisieren zu lassen, wird leider oft die Basisapplikation immer komplexer, so dass sich viele einfach bedienbare Branchenlösungen immer mehr zum komplexen Funktionselefanten entwickelt haben. Das führt dazu, dass Kunden weiter die veraltete Version der Software einsetzen.

Der übliche Ansatz in mindestens 80% aller Delphi Softwarehäuser ist eine komplexe Anwendung mit einem zentralen Delphi Quellcode und einer zentralen Datenbankstruktur. Diese muss für alle Kunden immer gleich sein, weil man keinen funktionierenden Variantenmechanismus hat. Die übrigen 20% nutzen meistens Techniken wie Packages, um jedenfalls in gewissen Teilen individuelle Kundenwünsche erfüllen zu können. Die Abhängigkeiten sind dabei jedoch nur unwesentlich weniger komplex als bei Compilerschaltern oder anderen Ansätzen.

Reengineering und Versionswechsel

Leider ändern sich sowohl die Kunden als auch die Prozesse, so dass etliche Softwarehäuser bereits an etwaigen Anforderungen scheitern. Mit dem Umstieg auf Delphi 2009 wurde die Unicodefähigkeit zwar eingeführt, aber einige ältere Komponenten waren leider inkompatibel. Zu diesem Zeitpunkt war auch bereits der Markt der Komponentenhersteller deutlich eingebrochen, so dass viele Komponenten gar nicht mehr aktualisiert wurden und so in der neuen Delphi Umgebung nicht benutzbar waren.

Aus diesem Grund stecken ganze Projekte auch heute noch fest. Diese Projekte können auch in absehbarer Zeit nicht auf neue Delphi Versionen umgestellt werden, weil zahlreiche Komponenten im gesamten Quellcode verknüpft sind. Ein Ersatz einer alten, etablierten Komponentenbibliothek erfordert für viele Softwarehäuser eine erhebliche Einschränkung oder sogar eine komplett andere GUI.

Die Frage stellt sich in so einem Fall für jedes Softwarehaus, ob es sich rechnet, mehrere Mannjahre in die Entwicklung einer neuen Version auf Basis einer neuen Delphi Version zu investieren, um am Ende vielleicht viele Kunden mit der dann zwangsweise eingeführten neuen GUI zu verärgern.

Selbst ein Unternehmen wie Microsoft hat bei Office und aktuell bei Windows 8 gerade unter langjährigen Anwendern viele Freunde verloren, die sich nun auf ältere Versionen beschränken oder ganz zu Alternativen gewechselt sind.

Gerade hier beginnen aber doch aus meiner Sicht die Vorwürfe und das Unverständnis gegenüber Borland bzw. Embarcadero. Die neuen Delphi Versionen versuchen jede noch so banale Komponentenbibliothek inkompatibel zu machen, auch wenn diese eigentlich 100% kompatibel zu neueren Versionen sind. Es werden immer wieder künstliche Hürden aufgebaut, die es verhindern, vorhandene Projekte mit allen Komponenten in einer neuen Delphi Version zu benutzen. Ob es Compilerversionsnummern sind, geänderte Units, geänderte Aufrufinterfaces oder Typen. Selbst bei banalen Projekten mit Fremdkomponenten ist der Aufwand der Konvertierung nicht zu unterschätzen. Seit Delphi nun nahezu halbjährlich als neue Version erscheint, ist der Aufwand, sein Projekt in der jeweils neuesten Version überhaupt auch nur kompilieren zu können, enorm.

Softwarearchitektur Teil 2

Vor vier Jahren haben wir für einen sehr innovativ denkenden Kunden aus der metall- und kunststoff-verarbeitenden Branche ein neues ERP Projekt begonnen, da unser Kunde sich mit den am Markt existierenden Lösungen nicht anfreunden konnte. Das hatte verschiedene Gründe. In erster Linie waren die meisten Lösungen zu kompliziert zu bedienen, zu teuer und in vielen Bereichen zu starr.

Da unser neues Projekt unter dem Namen BRP-Software auch als Grundlage für weitere Kundenprojekte und auch für unsere interne Auftragsbearbeitung und Lizenzverwaltung zum Einsatz kommen sollte, waren viele Punkte von vornherein klar:

- Der Client wird als Vorbild eher einem Webbrowser entsprechen, der anhand eines geeigneten Regelwerks die Inhalte der Datenbank individuell darstellt.
- Die Business Logik hat nichts im Client zu suchen, sondern kommt in die Firebird Datenbank.
- Das Datenmodell basiert auf dem Einsatz bestimmter, selbsterstellter Systemtabellen, aus denen der Client flexibel die notwendigen Informationen bezieht, um die Datenmasken zu erstellen.
- Der gleiche Client kann für verschiedene Datenbanken benutzt werden und stellt dann, je nach Datenbankinhalt, die Eingabeformulare auf dem Bildschirm dar.
- Der Client soll ohne Tricksereien plattformübergreifend nativ einsetzbar sein. Plattformen sind dabei 32 und 64 Bit Windows, Linux und Mac OS-X.
- Eine native Unterstützung für Firebird als Datenbank war für alle Plattformen wichtig.
- Entwicklung und Testen muss nahezu gleichzeitig möglich sein.
- Es gibt kein Pflichtenheft, sondern sämtliche Businessprozesse werden nach Bedarf entworfen und im System am gleichen Tag getestet und ggf. in das Echtsystem übernommen.

- Komponenten werden nur dann benutzt, wenn diese bereits in der IDE auf allen Plattformen verfügbar sind.
- Grafische Funktionen werden bevorzugt durch eigene OwnerDraw Routinen implementiert.

Die damals aktuelle Delphi Version 2010 und auch die neue XE Version hatten dabei noch diverse fehlende Funktionen, so dass wir uns nach diversen Tests für Lazarus entschieden haben. Beide Delphi Versionen waren noch reine Win32 Plattformen, obwohl win64 Rechner schon weit verbreitet und eine x64 Delphi Version schon lange angekündigt war. Wir hatten bereits erste positive Erfahrungen mit Lazarus in kleineren Kundenprojekten gesammelt, insbesondere mit 64 Bit Anwendungen.

Für unser neues Produkt mit dem Namen BRP-Software haben wir nun noch einmal sämtliche Erfahrungen der letzten Jahre zusammengefasst. Ganz besonders wichtig dabei war uns die Auswahl der IDE als auch die Auswahl eventuell benutzbarer Komponenten. Denn keineswegs wollten wir die Fehler machen, die wir seit Jahren bei Kundenprojekten aufgezeigt haben.

Gleichzeitig war aus meiner Sicht ein integraler Bestandteil der neuen Software die Fähigkeit, wie ein Webbrowser zu arbeiten und als Gegenseite von einem Firebird Server bedient zu werden, der dabei sowohl das Datenmodell vorgibt als auch die Business Logik und Prozeduren. Sämtliche visuellen Komponenten werden dynamisch erzeugt.

Daher befinden sich in der IBExpert Kundendatenbank völlig andere Tabellen und Prozesse als in der Datenbank unserer Kunden. Während für uns die Lizensierung ein wichtiger Bestandteil ist, haben unsere Kunden damit überhaupt nichts zu tun. Diese Tabelle existiert also nicht in deren Datenbank. Der Artikelstamm im Hause IBExpert unterscheidet sich auch erheblich vom Artikelstamm, den wir für ein individuelles Kundenprojekt im Logistikbereich umgesetzt haben.

Sobald der Anwender die BRP-Software startet, wird in der verbundenen Datenbank der Benutzer mit seinen Rechten geprüft und angezeigt, was für diesen Anwender freigeschaltet ist. Sobald der Anwender nun in den Bereich Artikelverwaltung wechselt, wird nun je nach Tabellenaufbau und Metadaten die Maske passend zur Datenbanktabelle angezeigt.

Wir oder die Kunden selbst können individuelle Anpassungen an der Datenbank erstellen. Der BRP Browser berücksichtigt diese Anpassungen sofort.

Bei der BRP Enterprise Version wurde in Zusammenarbeit mit einem Kunden zum Beispiel ein sehr flexibles Arbeitszeiterfassungsmodell umgesetzt. Die Akzeptanz unter den Mitarbeitern ist sehr hoch. Die Daten werden sehr genau den wirklichen Arbeitszeiten entsprechend erfasst, so dass auch die Fertigungsplanung und Nachkalkulation laufend exakte Daten über den Fertigungsstand vorliegen hat.

Für andere Kunden ist das eventuell viel zu komplex oder sogar noch zu einfach. Wir oder die Kunden selbst können individuelle Anpassungen an das System vornehmen.

Multiplattform

Entscheidend bei unserer Auswahl der Lazarus IDE ist die Tatsache, dass wir mit einer ganz klaren Architektur eine passende Programmiersprache gewählt haben, mit der wir diese Architektur auf den notwendigen Plattformen umsetzen können. Wir hätten sicherlich auch mit .NET basierenden Sprachen eine ähnliche Lösung erstellen können. Dabei ist aber dann der große Erfahrungsschatz aus mittlerweile 20

Jahren Delphi/Lazarus/VCL/Pascal Sprache nur begrenzt hilfreich. Hinzu kommt, dass .NET nur innerhalb der Windows Plattform auch Multiplattform ist.

Die Anzahl der mittlerweile gefundenen Sicherheitslücken in Java überzeugen mich jedes Mal wieder davon, auf eine gute Plattform gesetzt zu haben.

In der aktuellen Delphi Version geht Multiplattform für die unterstützten Plattformen nur mit Firemonkey, was ich aus verschiedenen Gründen ablehne. Die kritische Frage, der sich Embarcadero stellen muss, ist, warum die eigene IDE nicht auf einer anderen als der Win32 API zu realisieren ist. Da stimmt doch etwas im Konzept scheinbar nicht.

Lazarus als Alternative

Wer mit der nativen Lazarus auf Linux oder auf Mac OSX oder sogar auf dem Raspberry Pi entwickelt hat, wird wissen, warum Multiplattform GUI Entwicklung auch nur mit nativen IDEs auf den jeweiligen Plattformen sinnvoll ist. Den Hype um Mobiltelefon Applikationen kann ich nur begrenzt nachvollziehen, jedenfalls unter der Maßgabe, damit Geld verdienen zu wollen. Wir setzen auf Webapplikationen, die auch auf Windows Mobile und sonstigen Plattformen einsetzbar sind. Mit jQuery Mobile, ca. 30 Zeilen PHP und einem Apache/Firebird Server lassen sich damit ebenfalls sehr effektiv Datenbankanwendungen für die mobile Nutzung programmieren, die auch mit mehreren tausend Benutzern stabil laufen und aktuell sind. Leider zeigt sich z.B. auch Apple mit seinen Consumer-Anwendungen eher anwenderunfreundlich.

Unsere Kompatibilitätsprobleme lassen uns bei der IBExpert Entwicklung bis heute noch an Delphi 5 festhalten, weil eine Portierung der 1,7 Millionen Zeilen Quellcode auf eine neuere Delphi Version aus unserer Sicht eher einer kompletten Neuentwicklung gleichkommt, ohne dadurch nennenswerte Vorteile zu erzielen. Wir kämpfen selbst heute noch mit der sehr auf die damalige Delphi Version und der Komponenten zugeschnittenen globalen Architektur, mit der wir IBExpert im Jahr 1999 begonnen haben. Das Projekt ist als klassisches Delphi Projekt mit vielen Formularen und Units entstanden und in vielen Bereichen sehr komplex miteinander verschachtelt.

Sollten wir mit Delphi 5 irgendwann doch wirkliche Grenzen erreichen, werden wir ganz sicher einen Nachfolger auf Basis von Lazarus entwickeln, um große Teile des vorhandenen Quellcodes weiterhin nutzen zu können. Dass uns mit dem Einsatz von Lazarus auch noch die gesamte IDE als Quellcode vorliegt und wir ggf. ärgerliche Fehler in der IDE selbst beheben können, ist dabei ein weiterer Vorteil.

Warum nicht jedes Delphi Projekt scheitert?

In jeder Programmiersprache und den meisten IDEs können gute oder schlechte Ergebnisse erreicht werden. Wobei gut dabei übrigens nicht unbedingt schön heißt. Aus betriebswirtschaftlicher Sicht ist der Faktor Produktivität wesentlich wichtiger. Wenn ich für ein Projekt vom Kunden 10 Manntage bezahlt bekomme, das Projekt aber in 5 Manntagen umsetzen kann, ist das auf jeden Fall gut, wenn ich 20 Manntage benötige, eher schlecht. Wenn ich bei der Preisfindung den Aufwand gar nicht abschätzen, sondern eigentlich nur raten kann, wie viel Zeit ich für das Projekt benötige und das Angebot darauf abstimme, dann ist diese Vorgehensweise selten langfristig erfolgreich.

Der Vorteil und gleichzeitig der größte Nachteil einer Delphi Entwicklung ist, dass auch ohne große Architektur eine Software erstellt werden kann, die durchaus die Anwenderseite befriedigt. Diese ist aber dann oft aus der Sicht des Programmierers sehr ineffektiv realisiert. Man kann sich aber auch in der

Architektur komplett verlieren, ohne dass man irgendwelche benutzbaren Ergebnisse liefert. Ein Quellcode mit intensiver Nutzung von Generics und anderen Spezifikationen ist nicht per se besser als einer ohne. Ein schlechter Quelltext wird auch durch Generics oder andere Techniken selten besser.

Das ist auch in Lazarus nicht anders. Der größte Vorteil bei Lazarus ist die dauerhafte Kompatibilität zu älteren Versionen und zu anderen Plattformen. Wer sich nicht ständig mit der Neuinstallation der IDE beschäftigen möchte, weil Bugs nur noch in neuen, kostenpflichtigen Versionen behoben werden, wird Lazarus schätzen. Auch deswegen, weil Lazarus auch jederzeit von einer Wechselfestplatte oder einem USB Stick betrieben werden kann, denn sämtliche Pfade sind nicht irgendwo in der Registry hinterlegt. Suchpfade zu neuen Komponenten werden über die LPK-Pakete automatisch erkannt.

Meiner Meinung nach haben schätzungsweise maximal 10% der Softwarehersteller, die auf Delphi setzen, eine flexible und leistungsfähige Architektur im Einsatz, mit der auch alle Entwickler zufrieden sind. Die übrigen 90% befinden sich mehr oder weniger weit in einer Sackgasse, weil grundlegende Abhängigkeiten echte Innovationen verhindern. Bei vielen Kunden sieht man nur einen Umstieg auf Visual Studio als möglichen Ausweg. Das ist aber keineswegs die einzige Option, sondern beinhaltet in diesem Falle immer eine komplette Neuentwicklung. Diese wird ohne geeignete Architektur in 5 bis 10 Jahren wieder in der gleichen Sackgasse enden.

Die Einführung einer echten und flexiblen Architektur in der Entwicklung ist sowohl auf Basis von der bereits benutzten Delphi Version, als auch auf Basis von Lazarus oder anderen Plattformen langfristig wesentlich erfolgversprechender, als der Umstieg auf eine andere Plattform, für die primär das komplette Know-How aufgebaut werden muss. Wir helfen gerne dabei.

Die BRP Technik in der Praxis

Melden Sie sich einfach bei uns unter info@ibexpert.com oder telefonisch unter +49 4408 3593492. Wir vereinbaren dann einen Termin und zeigen Ihnen kostenlos und unverbindlich in einer Fernwartungs-sitzung von 60 bis 120 Minuten, wie unser System funktioniert und wie Sie das System selbst programmieren können.

Für die Nutzung unserer Technologie bieten wir unterschiedliche Lizenzmodelle, geeignet für die Anwendung der kompletten Fertigungs- und ERP Software im eigenen Haus, aber auch für Systemhäuser, die eine Branchenlösung darauf aufbauen möchten. Alles, was Sie für die eigenen Anpassungen benötigen, sind grundlegende Firebird SQL Kenntnisse.