kleiner kommunikation
max@kleiner.com

1 MDD mit ECO II

Wie man mit Delphi 2005 und ECO II eine persistente und objektorientierte Anwendung aufbaut, will ich im vorliegenden Bericht modellgetrieben aufzeigen. Für Geschäftsobjekte oder datenintensive Systeme sind die Automatismen der MDD (model driven development) in ECO eine enorme Unterstützung, zumal man auch ASP.net Seiten oder Web Services damit generieren kann.

1.1 Einstieg mit Klasse

ECO (steht für Enterprise Core Objects) ermöglicht die Implementierung und Steuerung einer MVC (Model View Controller) Architektur, d.h. eine Mittel-Schicht von Kontrollklassen (Presentation, Mapping und Handles) vermittelt zwischen den Geschäftsobjekten und der graphischen Repräsentation. Zusätzlich bietet ECO ein durchdachtes Datenbankhandling an, das nebst dem «Klassen zu Tabellen Mapping» in eine relationale Datenbank künftig auch Cached Updates sowie Transaction Handling erlaubt.

Das direkte Einbinden der Business-Objekte ist wohl der größte Vorteil von ECO. Der Code für die Geschäftsobjekte lässt sich größtenteils automatisieren und auch bei Änderungen oder Erweiterungen durchgängig aktualisieren. Hier muss aber die Richtung des Programmiermodells eingehalten werden, d.h. Anpassung oder Erweiterungen erfolgen stets in der Modelloberfläche. ECO erlaubt bei bestehenden Datenmodellen aber auch ein Reverse Engineering und generiert aus der DB die Klassen- und Packetstruktur.

Für die Veteranen unter uns sei erwähnt, dass die Entwickler von ECO mit dem früheren Bold identisch sind und seit dem 4. Mai auch Bold für Delphi 2005 inklusive Quellen kostenlos anbieten, setzt aber den Architekten voraus:

http://bdn.borland.com/article/0,1410,33058,00.html

Das Einbinden der zum Download freigegebenen Packages verlief zügig, einziger Wermutstropfen, die Weiterentwicklung von Bold wird eingestellt, hier setzt Borland nun auf ECO.

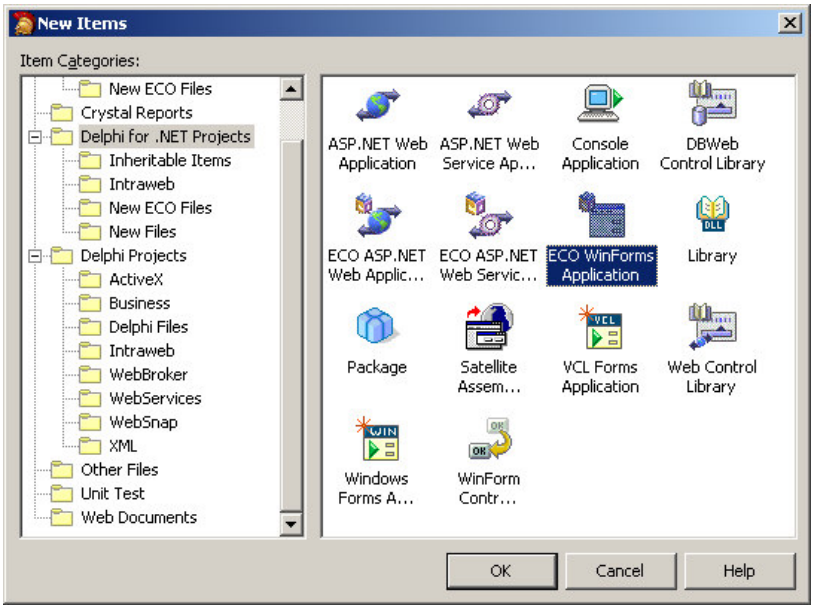


Abb. 1: Start aus der Projektgalerie

Starten wir mit dem Praktikum. Mit Delphi 2005 wählen wir FILE | NEW - OTHER und dann in “Delphi for .NET Projects” die ECO „WinForms Application“ aus. ECO ist demzufolge nur für .NET Projekte bestimmt, für Win32 gibt es ja das ausgereifte Bold.

Nach dem Eingeben des Applikationsnamens „EcoLoco“, die eine einfache Modellbahnsteuerung vorwegnimmt, baut Delphi das Grundgerüst inklusive Ressourcen- und Projektdatei zusammen. Man kann vor allem, wie in Abb. 2 ersichtlich, vier generierte Dateien entdecken:

- *CoreClasses.txvpck*, *CoreClassesUnit.pas*, *EcoLocoEcoSpace.pas* und *WinForm1.pas*

Die erste Datei CoreClasses stellt mit CoreClassesUnit ein leeres UML-Paket dar, welches ich nun gemäss Abb. 1 mit 3 Klassen anreichere. Dazu wechsle ich rechts auf das Register Modellansicht und öffne den Untereintrag CoreClasses damit die Modelloberfläche erscheint. Wenn das nicht klappt, versuchen Sie mal zu kompilieren um dann in die aktivierte Modelloberfläche zu gelangen. Nach dem Erstellen der 3 Klassen mit Hilfe der Together Toolpalette und dem UML Designer setze ich im Objektinspektor bei der Klasse `TLocomotive` noch die Eigenschaft Abstract auf `True`.
Soweit ist das Modell erstellt und im Hintergrund werden die zugehörigen Units aktualisiert.

```
ecoLocoEcoSpace in 'ecoLocoEcoSpace.pas'
  {ecoLocoEcoSpace.TecoLocoEcoSpace: Borland.Eco.Handles.DefaultEcoSpace},
  CoreClassesUnit in 'CoreClassesUnit.pas',
  WinForm1 in 'WinForm1.pas' {WinForm1.TWinForm1:
System.Windows.Forms.Form};
```

Die nun erzeugten Codedateien werden in den Delphi Editor geladen. Diese Dateien sollte man eigentlich nicht ändern, da sie der Generator bei der nächsten Modelländerung überschreiben wird. Allerdings ist die Sache differenzierter, da ECO präzisiert, wo in einer Unit keine manuelle Änderung erfolgen sollte. Werfen wir einen Blick in die Datei *CoreClassesUnit.pas*.

Die Attribute `[EcoAutoGenerated]` oder `[EcoAutoMaintained]` sorgen für diese Flexibilisierung bei nachträglich manuellen Erweiterungen:

```
  [UmlElement]
  [UmlMetaAttribute('stereotype', 'abstract_factory')]
  TLocomotive = class abstract(System.Object, ILoopBack)
  strict protected
    [EcoAutoGenerated]
    _content: IContent;
    function get__loctype: string;
    procedure set__loctype(Value: string);
    [EcoAutoGenerated]
    property _loctype: string read get__loctype write set__loctype;
    [EcoAutoGenerated]
    procedure Initialize(serviceProvider: IEcoServiceProvider);
    [EcoAutoGenerated]
    procedure Deinitialize(serviceProvider: IEcoServiceProvider);
    function get__status: string;
    procedure set__status(Value: string);type
```

Die generierten Felder der Klasse werden mit getter- und setter Methoden so ausgerichtet, daß man auf die Objekte möglichst effizient und sicher darauf zugreifen kann. Auf der anderen Seite erhöhen Sie mit Code-Generatoren die Abhängigkeit. Ein typisches Optimierungsproblem zwischen „Automatismus und Flexibilität, welches der Entwickler eines modellgetriebenen Systems mit berücksichtigen sollte.
Die abstrakte Klasse `TLocomotive` ist nun in der Lage, die Struktur hinsichtlich Persistenz und Zugriff den Unterklassen anzubieten, zudem Mechanismen zur Abfrage, Navigierbarkeit und Darstellung bereitzustellen, dies bis anhin ohne eine Zeile programmiert zu haben.

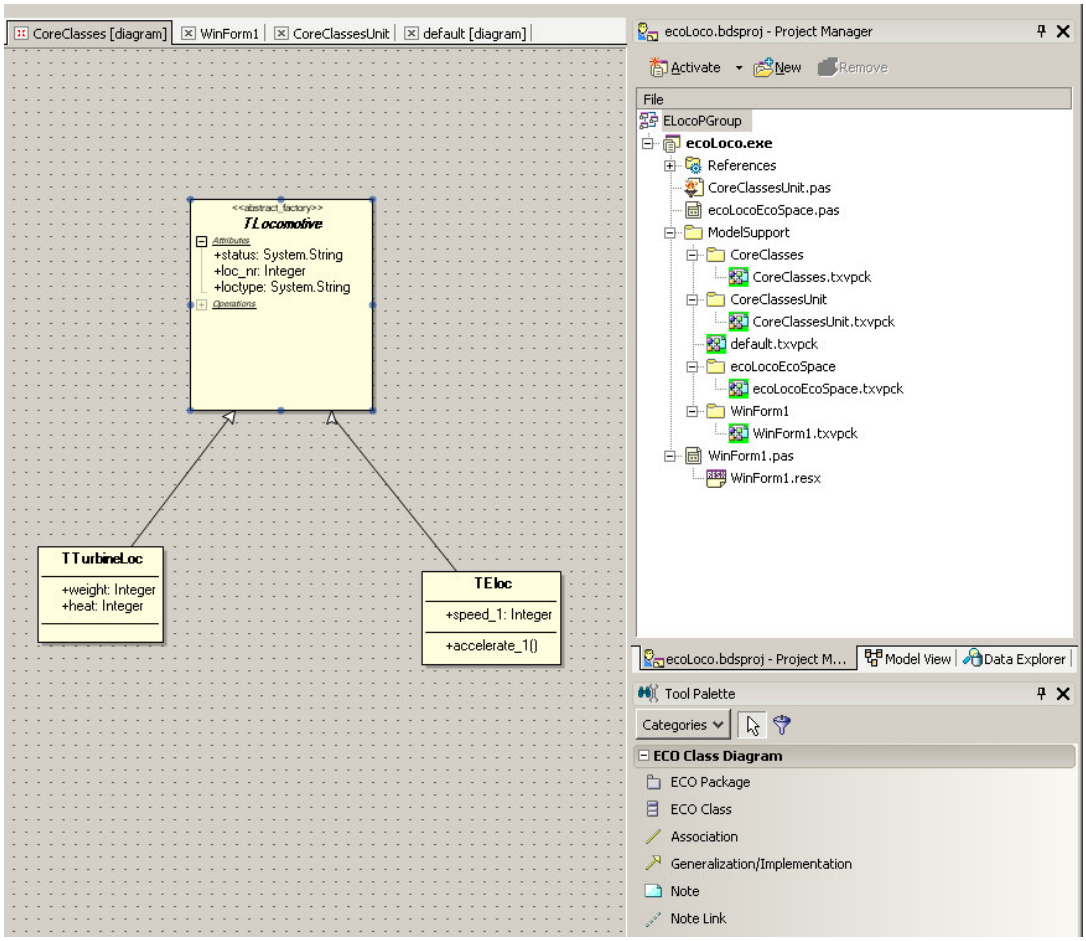
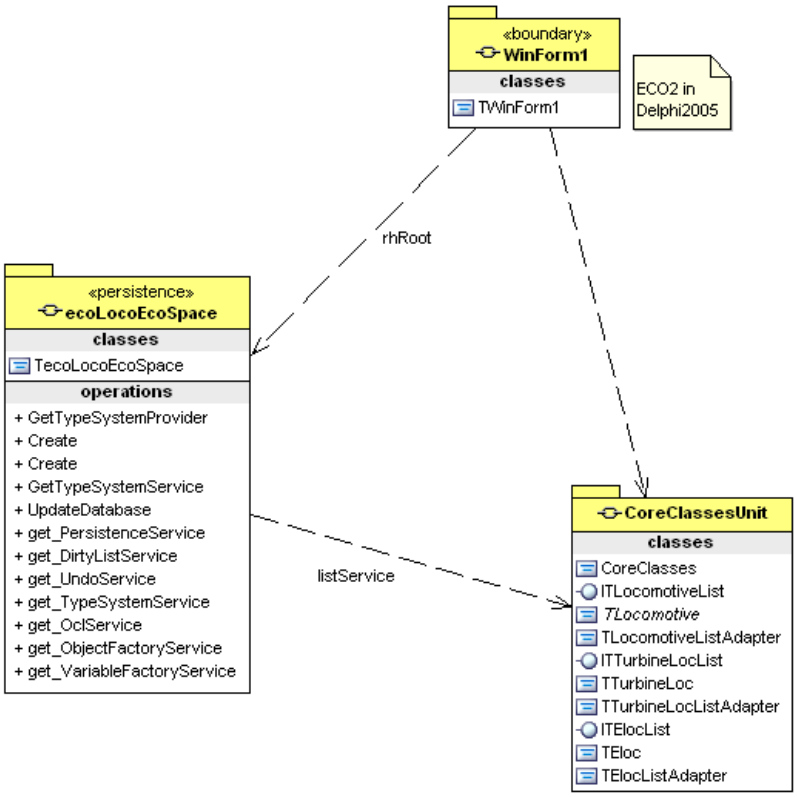


Abb. 2: Die Modelloberfläche von Together

Auch interessant sind in der generierten `CoreClassesUnit`, wie Abb. 3 zeigt, die Listen und Adapter. Im Zuge der Tabellen- oder XML-Generierung kann ECO auch die Codegenerierung der Hilfsklassen bewerkstelligen. Das Framework erzeugt Code, den man im aktiven Modell nicht explizit darstellen kann. Konkret werden ja nebst den eigentlichen Geschäftsobjekten noch Listen- und Adapter-Klassen generiert, da ECO für jede erzeugte Klasse eine typgerechte Liste benötigt.

Mit diesen Objektlisten kommt der volle Nutzen der strikten Typüberprüfung des Compilers zum Tragen. Bei benötigten Assoziationen zwischen Objekten legt ECO auch die zugehörigen Linkklassen an, weil wir bspw. eine n:m Relation in der DB abbilden wollen. Eine Datenbank oder eine XML Datei wird aus den Klassen ja automatisch generiert und lässt sich später modifizieren, wenn das Modell eine Änderung erfährt.



// pac_Eloco.png

Abb. 3: Überblick im Paketdiagramm

Diese Adapter sind zudem verantwortlich für die Implementierung der Interfaces inklusive der Initialisierung, andererseits wird die Typenkompatibilität erfüllt, da ECO ja mit typisierten Listen arbeitet.

```
function TLocomotive.TLocomotiveListAdapter.Add(value: TLocomotive):
                                                              Integer;
begin
  Result:= Self.Adaptee.Add(value);
end;
```

1.2 Der ECO Space

Das von mir erstellte Modell, enthält die nötigen Informationen zur automatischen Generierung einer XML Datei oder einer Datenbank mit den zugehörigen Tabellen. Dieses Klassen zu Tabellen Mapping lässt sich nach der Generierung weiter steuern, da in der Regel die gewünschte Datenbank mit Indizes, Triggern oder Constraints erweitert wird. ECO kann in diesem Sinne nur die Client-seitigen Rules beeinflussen. Der Schemagenerator ist betreffend Änderungen des Modells vorbereitet, mit OCL auch Server-seitige Regeln zu beeinflussen.

Als nächstes kommt der EcoSpace zum Zuge, der mir die erwähnte XML-Datei generiert und somit meine Objekte „lagerfähig“ macht. Nun, was ist der EcoSpace? Dieser Raum beinhaltet einen Container von Objekten, der zur Laufzeit die aktuellen Instanzen und Relationen zurückgibt. Anders ausgedrückt kann man den EcoSpace als aktive Instanz des Modells (UML Packages) betrachten. Die zugehörigen Objekte inklusive Methoden, Eigenschaften und Assoziationen entsprechen anstandslos der Typ-Domäne. Da der EcoSpace wie ein Cache funktioniert und auch die Transaktionen im Kontext steuert, muss er wissen wie die Speicherung erfolgt. Dies erreiche ich mit der Komponente `PersistenceMapper`, die ich nun über die IDE konfigurieren will.

In der Projektverwaltung doppelklicke ich auf `ecoLocoEcoSpace`, um den Design Editor aufzurufen. Sie werden nun, so hoffe ich doch, staunen wie leicht die Speicherfähigkeit der Objekte (Persistenz)

erreicht wird. Aus der Toolpalette wähle ich die Komponente `PersistenceMapperXml` welche aus dem Namensraum `Borland.Eco.Persistence.` stammt. In der Eigenschaft `FileName` setze ich *locomotiveStore.xml* ein und verknüpfe das Objekt `PersistenceMapperELocoXml1` mit dem EcoSpace innerhalb der Eigenschaft `PersistenceMapper`. Generiert wird folgender Code:

```
procedure TecoLocoEcoSpace.InitializeComponent;
begin
  Self.PersistenceMapperELocoXml1 :=
Borland.Eco.Persistence.PersistenceMapperXml.Create;
  Self.PersistenceMapperELocoXml1.FileName := '\locomotivestore2.xml';
  Self.PersistenceMapper := Self.PersistenceMapperELocoXml1;
end;
```

Damit ist der grundlegende Rahmen unseres Code geschaffen. Das Zulassen von Abonnements wird in ECO mit den Subscriptions gelöst, welche bei einer Wertänderung Benachrichtigungen auf der Oberfläche ermöglichen. Mit dieser Technik ist auch die Unterstützung eines Optimistic Locking einer DB mit Konfliktauflösung möglich.

Ich wechsle nun zur WinForm und füge gemäss Abb. 4 die DataGrids, ExpressionHandles und die weiteren Steuerelemente hinzu. Bei den 3 ExpressionHandles setze ich die Eigenschaft `RootHandle` jeweils auf `rhRoot` und bei der Eigenschaft `Expression` gebe ich im OCL-Editor je `TEloc.allInstances`, `TLocomotive.allInstances` und `TTurbineLoc.allInstances` ein. Die zugehörigen DataGrids lassen sich mit der Eigenschaft `DataSource` auf die ExpressionHandles verknüpfen. ExpressionHandles lassen sich allgemein zum Evaluieren von OCL Abfragen einsetzen, hier erhalten ich die Ergebnismenge aller Instanzen einer ECO Klasse zurück.
Beachten Sie unterhalb dem WinForm Designer die nicht visuelle Komponente `rhRoot`. Dieses Reference Handle ist die eigentliche und einzige Verbindung zwischen den diversen Forms (WinForm, WebForm etc.) und dem EcoSpace mit den darin enthaltenen Objekten, wie auch der folgende Konstruktor verdeutlicht:

```
constructor TWinForm1.Create;
begin
  inherited Create;
  InitializeComponent;
  FEcoSpace:=  TecoLocoEcoSpace.Create;
  rhRoot.EcoSpace:= FEcoSpace;
  FEcoSpace.Active:= True;
end;
```

Als nächstes benötige ich das geeignete Ereignis, um die Lokomotiven als Objekt im Grid aufzunehmen. Es ist natürlich der Standardklick, der mit folgendem Code gefüttert wird. Nach der Instanzierung habe ich den gewohnten Zugriff auf Methoden und Eigenschaften.

```
procedure TWinForm1.btnAddEloc_Click(sender: System.Object; e:
System.EventArgs);
var eloc: TEloc;
begin
  eloc:= TEloc.Create(EcoSpace);
  eloc.status:= 'track_12_experimental';
end;
```

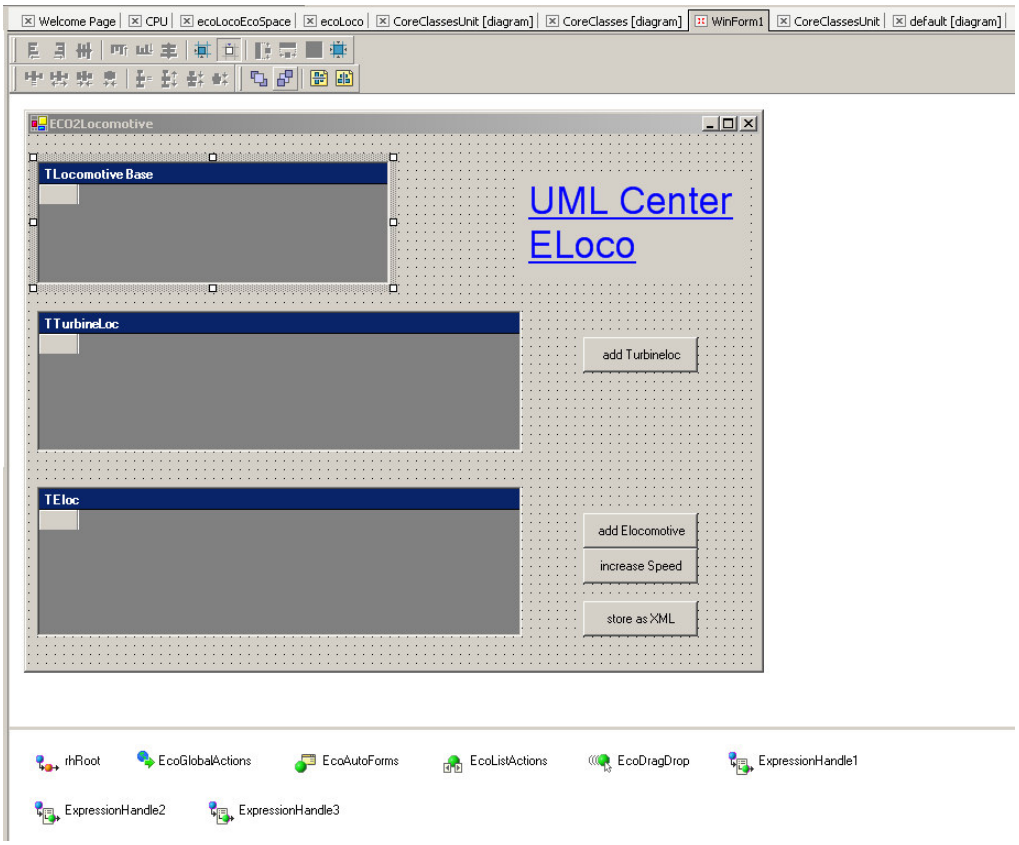


Abb. 4: Die GUI der “Modellloco”

Schlussendlich muss noch ein Knopf her, welcher das Zurückschreiben erlaubt. Die Technik der Serialisierung kommt generell aus dem Namensraum:

```
using System.Runtime.Serialization.Formatters.Soap;

procedure TWinForm1.btnStorexml_Click(sender: System.Object; e:
System.EventArgs);
begin
  ecoSpace.UpdateDatabase
end;
```

Als letztes Beispiel sei die Möglichkeit erwähnt, das Modell mit Methoden zu erweitern. In meinem Fall geben wir der ELok mehr Schub indem wir aus der Modelloberfläche in Abb.2 die Methode `accelerate()` generieren lassen und den Eventhandler mit folgendem Code bestücken:

```
procedure TWinForm1.btnSpeed_Click(sender: System.Object;
                                                  e: System.EventArgs);
var eloc: TEloc;
begin
// cmlocos connects to ehelocs to be type safe
  try
    eloc:= (cmLocos.element.asObject as TEloc);
    eloc.accelerate_1(3);
  except
    on e: Exception do
      messageBox.Show(e.Message)
  end;
end;
```

Mit ECO ist MDA keine Leerformel mehr. Kernidee der MDA ist ja die schrittweise Verfeinerung von der Analyse zum Design ausgehend von einer Modellierung der fachlichen Applikationslogik, die unabhängig von der technischen Implementierung einer DB oder sonstigen Schnittstelle ist. Mit Delphi 2005 und ECO II habe ich konkret gezeigt, dass Modellieren gar nicht so abstrakt ist;)

Max Kleiner, Juli 2005

Links und Literatur:

http://www.borland.com/delphi_net/architect/eco/tutorial/
Kleiner et al., Patterns konkret, 2003, Software & Support

Sourcen:

http://www.softwareschule.ch/download/eco22.zip

Beispiel auf der CD-ROM