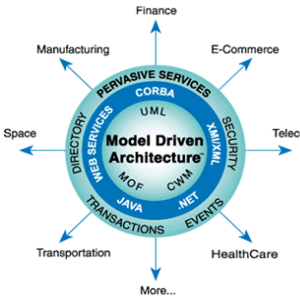
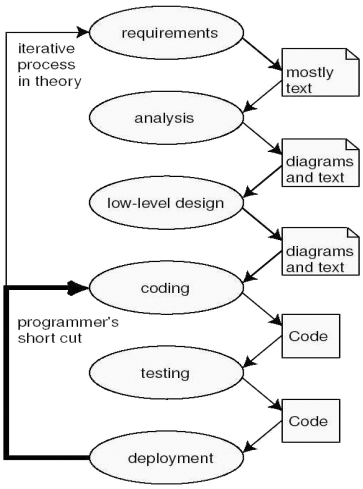
Dialogentwicklung mit Hilfe des Model Driven Architecture Ansatzes

Peter Mössinger, LRP Landesbank Rheinland-Pfalz



ollständig neue Möglichkeiten im Bereich der
iese neuen Wege werden exemplarisch am Beispiel die
ezeigt. Zur Umsetzung gelangt Delphi (ECO) von
er Windows zum Einsatz.

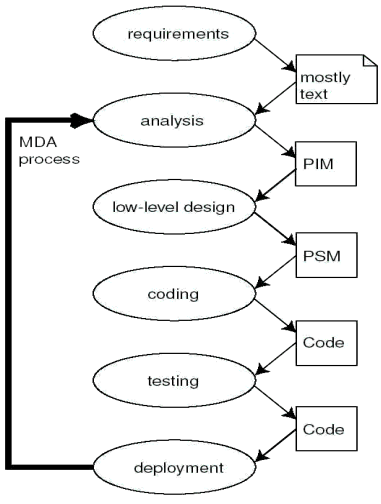
Der typische Software-Entwicklungs Prozess



Probleme dieses Prozesses:

- Kein Zusammenhang zwischen Spezifikationen bzw. Analysen und dem Code → Analysen werden als „nicht produktiv“/Overhead betrachtet
- Bei Korrekturen und Weiterentwicklungen wird die Spezifikation nicht angepasst → sie verliert an Wert

Der MDA Prozess



PIM: Plattform Independend Model

PSM: Plattform Specific Model

Voraussetzung des MDA-Prozesses:

Analyse (Model) und low-level Design werden in den Software-Entwicklungsprozess eingebunden

Model, low-level Design und Code automatisch synchronisiert

Problem: Wohldefinierte Darstellungsform (Sprache) für Model und low-level Design, freier Text reicht nicht aus

Automatische Transformationen zwischen den verschiedenen Sprachen

Neue Sprachen und Tools zur Erreichung dieses Ziels

Analyse:

Was ist ein Model?

A model is a description of (part of) a system written in a well defined-language.

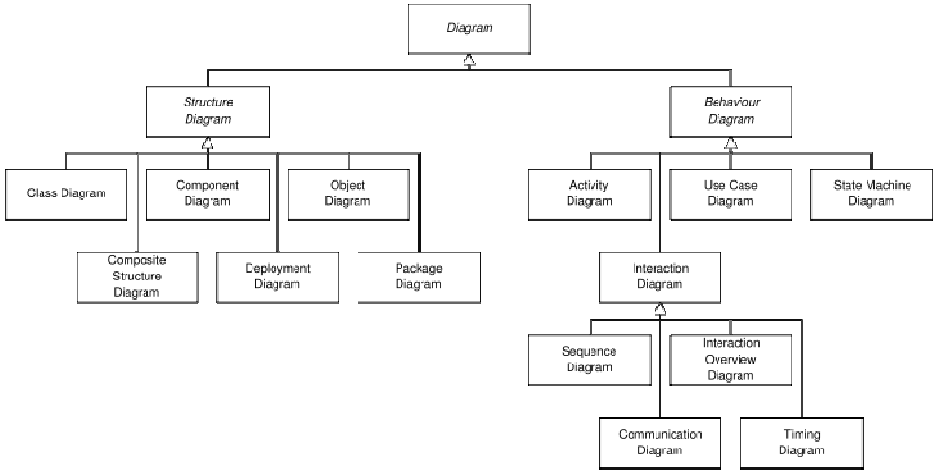
A well defined-language is a language in well-defined form (syntax), and meaning (semantics), which is suitable for automated interpretation by a computer. ([1])

PIM zur Beschreibung eines Systems ohne Festlegung auf Zielplattform

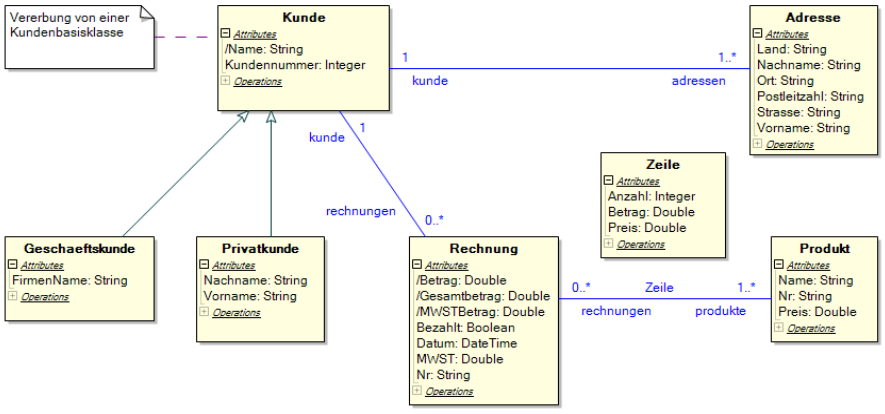
Meist genutzte Modellierungssprache: UML (Unified Modeling Language, Standard der OMG, Object Management Group, www.omg.org)

UML: Grafische Notationen für Modelle von statischen Strukturen und dynamischen Abläufen

Diagrammtypen in UML 2.0:



Beispiel eines PIM: Das Klassendiagramm



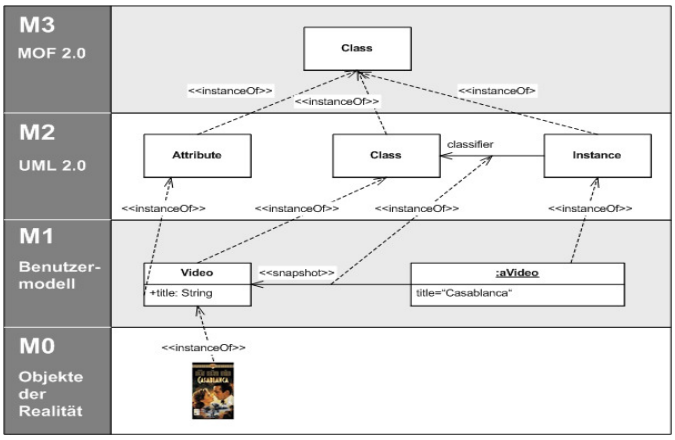
- *Klassisches UML: Nur statische Struktur-Informationen*
- *OCL (Object Constraint Language): Abfrage und Ausdrucks-Sprache für UML*
- *OCL-Einschränkungen und Ausdrücke:*

Produkt: self.Price>0
Zeile: self.Price>0
Rechnung: (self.MWST>0) and (self.MWST<1)
*Rechnung: MWSTBetrag=self.Betrag*self.MWST*
Rechnung: Betrag=Zeile.allInstances.Betrag->sum
Rechnung: Gesamtbetrag=self.Betrag+self.MWSTBetrag

- *OCL-Abfragen: Objektorientiertes Pendant zu SQL*

Teil der Anwendung, nicht des Modells
Kunde.allInstances
Kunde.allInstances->select(oclIsTypeOf(Privatkunde))
Kunde.allInstances->select(rechnungen->notEmpty)

- **PIM und PSM werden in einer allgemeinen Meta-Sprache dargestellt: MOF (Meta Object Facility).**



- **Transformationen zwischen den Modellen sind Programme in der Sprache MOF QVT (Query View Transform)**

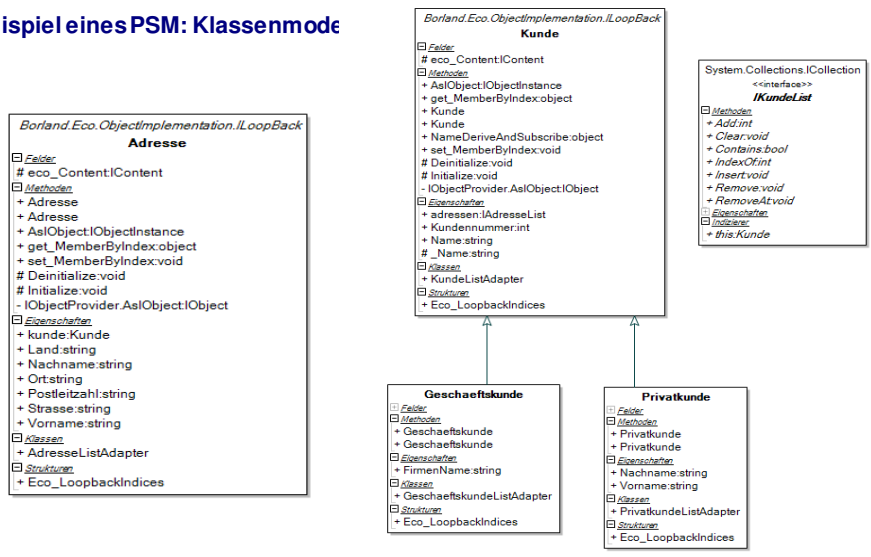
Transformation PIM → PSM

Aus einem PIM können mehrere PSMs entstehen.

Objektmodel für eine OO-Sprache
Entity-Relationship Modell für eine relationale Datenbank
usw.

Je nach Projekt werden unterschiedliche PSMs benötigt

Beispiel eines PSM: Klassenmode



Transformation PSM → Code

- Eigentliches Ziel der Modell-Transformationen
- Nicht nur Programmcode sondern auch Datenbankstruktur, User-Interface, ..
- Änderungen im Modell führen zu konsistenten Code-Änderungen
- 100% Code-Generierung nicht möglich: Manuell ergänzter Code muss bei Neugenerierung erhalten bleiben
- Hohe Codequalität durch automatische Generierung
- Fehler können durch Änderung des Generierungstools bereinigt werden

Vorteile:

- Sorgfältigere Konzeption in der Entwurfsphase
- Modelle entscheidend für Entwicklungsprozess → Modelle und Code bleiben synchron
- Standardisierte Codestruktur
- Bessere Wartbarkeit als manuell erzeugter Code, Umstellungen bzw. Technologiewechsel durch Änderung der Transformationen und Neugenerierung
- Plattform-Unabhängigkeit eines Teils des Systems → Unterstützung von Systemen, die über verschiedene Plattformen interoperieren
- Idealfall: Steigerung der Entwicklungsgeschwindigkeit, Entwicklungsprozess wird strukturierter, bessere Handhabbarkeit durch höhere Abstraktion

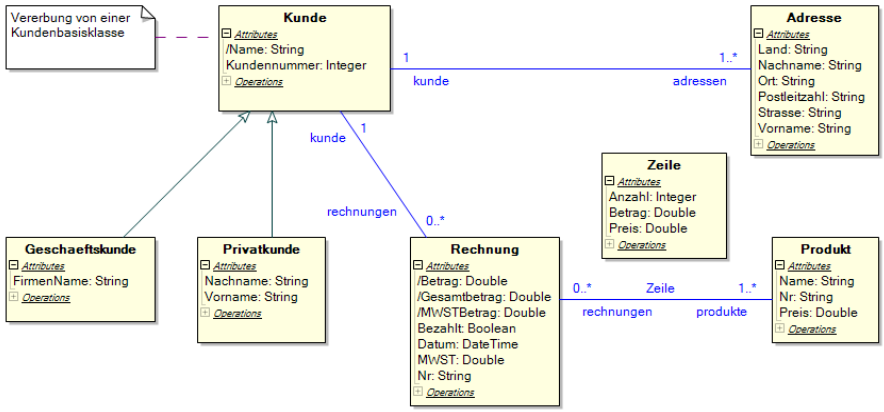
Nachteile:

- Erfordert deutlich höheres Know-How
- Höhere Kosten und Risiken wegen weiterer Tools (Langlebigkeit) speziell für kleinere Projekte
- Noch keine endgültige Standardisierung
- Nicht alles kann modelliert werden → verschiedene Teile des Systems werden unterschiedlich abgebildet
- Auswirkungen einer Neugenerierung auf ein laufendes System?

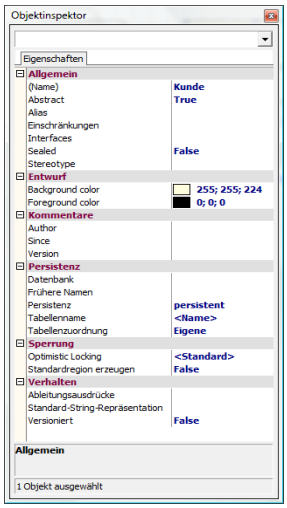
Enterprise Core Objects (ECO)

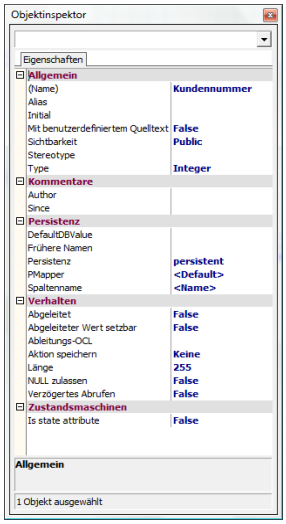
- Implementation des MDA Ansatzes von CodeGear (Borland)
- Bestandteil des Borland Developer Studios
- Unterstützte Konzepte: UML, OCL, Ausführbare State Machines
- „PIM“ mit Plattform-Informationen, PSM entfällt
- Direkte Code-Generierung aus dem „PIM“ (Live-Code)
- Vorgegebene Transformationen
- Programmiersprachen: C# und Delphi.NET
- Generierte Objekte werden in komfortables Framework eingebettet

ECO: Das Klassendiagramm einer Beispielanwendung



Eigenschaften von Klassen und Attributen:





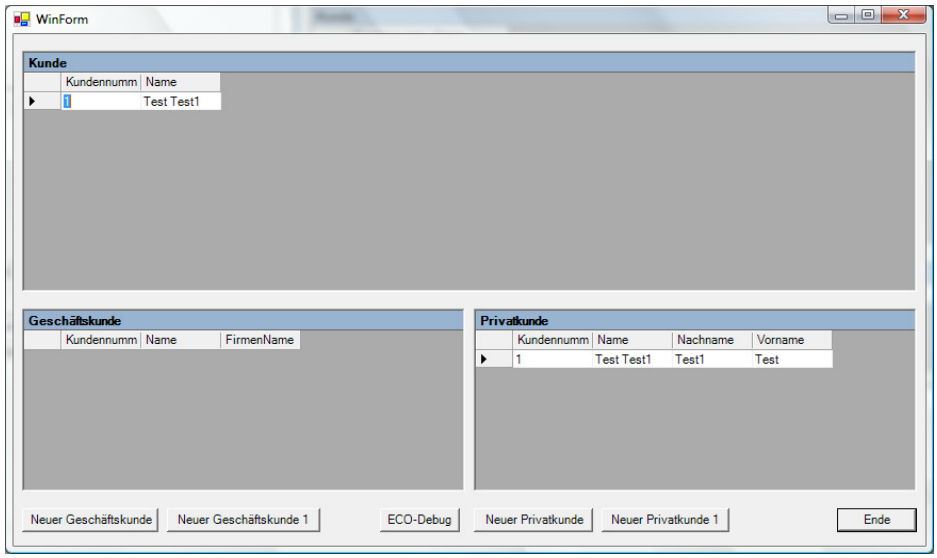
Erzeugter Code:

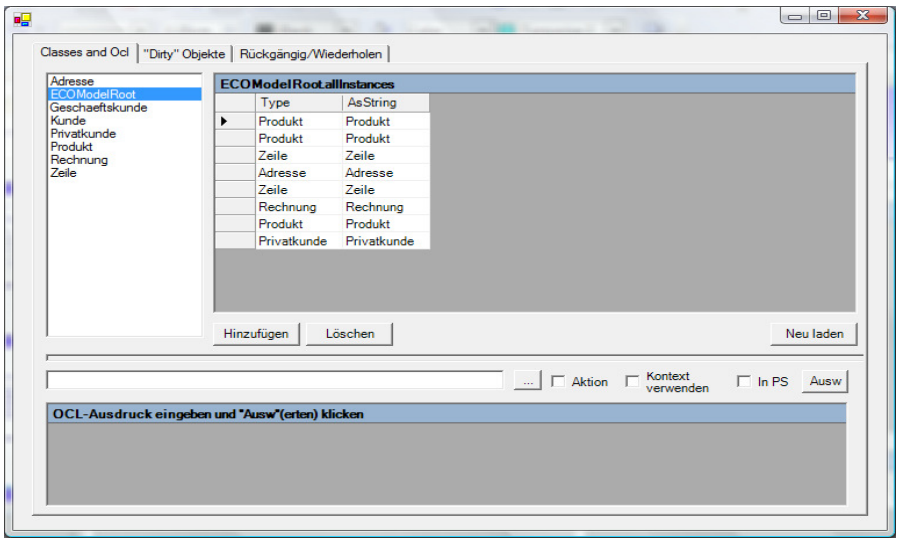
```
// Datei erstellt 04/05/2007 19:56:31
using System;
using System.Collections;
using Borland.Eco.Services;
using Borland.Eco.UmlRt;
using Borland.Eco.UmlCodeAttributes;
using Borland.Eco.ObjectImplementation;
using Borland.Eco.ObjectRepresentation;

namespace ECO_Beispiel
{
	[UmlElement(Id="0380f708-85dd-46f7-99f7-401ee739d6a4")]
	public class Kunde : ILoopBack
	{
		#region ECO generated code ...
		public Kunde(IEcoServiceProvider serviceProvider)
		{
			this.Initialize(serviceProvider);
			try
			{
				// Hier Benutzerquelltext
			}
			catch (System.Exception )
			{
				this.Deinitialize(serviceProvider);
				throw;
			}
		}
	...
}
```


ECO:

- Kapselt den Datenzugriff: Framework kümmert sich um die Persistenz
- Datenbank- und XML-Persistenz
- OR (Objekt-relationaler)-Mapper mit XML-Parametrisierung
- OCL als Abfragesprache
- Verschiedene Services unterstützen den Entwickler: Object Factory Service, OCL Service, Undo Service, Persistence Service, External ID Service, State Service, ...
- Unterstützung des .NET-Databindings
- Optimistic Locking
- Verzögertes Laden von Daten
- Automatische Dialoge mit Drag-and-Drop


WinForm
Kunde
Kundennumm
Name
1
Test Test1
Geschäftskunde
Kundennumm
Name
FirmenName
Privatkunde
Kundennumm
Name
Nachname
Vorname
1
Test Test1
Test1
Test
Neuer Geschäftskunde
Neuer Geschäftskunde 1
ECO-Debug
Neuer Privatkunde
Neuer Privatkunde 1
Ende


Classes and Ocl
"Dirty" Objekte
Rückgängig/Wiederholen
Adresse
ECOModelRoot
Geschaeftskunde
Kunde
Privatkunde
Produkt
Rechnung
Zeile
ECOModelRoot.allInstances
Type
AsString
Produkt
Produkt
Produkt
Produkt
Zeile
Zeile
Adresse
Adresse
Zeile
Zeile
Rechnung
Rechnung
Produkt
Produkt
Privatkunde
Privatkunde
Hinzufügen
Löschen
Neu laden
...
Aktion
Kontext verwenden
In PS
Ausw
OCL-Ausdruck eingeben und "Ausw"(erten) klicken

Literatur:

[1] MDA Explained: The Model Driven Architecture: Practice and Promise,
Wim. Bast, Anneke G. Kleppe, Jos B. Warmer, Addison-Wesley Professional
[2] http://www.omg.org/ + http://www.omg.org/mda/faq_mda.htm
[3] MDA Guide: http://www.omg.org/docs/omg/03-06-01.pdf
[4] http://de.wikipedia.org/wiki/Model_Driven_Architecture
[5] http://de.wikipedia.org/wiki/MDSD
[6] http://de.wikipedia.org/wiki/Meta-Object_Facility
[7] http://www.software-kompetenz.de/?26634
[8] http://info.borland.de/newsletter/nl04_4/Eco/Eco.htm
[9] http://en.wikipedia.org/wiki/CodeGear_ECO