

Pattern Praxis



Techniken lassen sich wiederholt anwenden:

- **Wieviele Substrings hat es im Wort Restaurant ?**
- **Wie zeichne ich aus einer römischen IX mit einem zusätzlichen Strich eine Sechs ?**
- **Was bedeutet: DER GEFANGENE FLOH**
- **Wie gross ist die W'keit 2 Vierer zu würfeln ?**
- **$P = U I$, $W = U I t$, welche Bedeutung hat die Formel ?**

Nutzen von Patterns

- Design or not sein --> Programming for change
- reuse --> Inheritance
- extendability --> Interfaces
- maintainability --> Modules
- availability --> Design by Contract
- useability --> Style Guides



Patterns History

- CH - Effekt: Pascal mit ersten Patterns (N. Wirth, 1971), WWW am CERN als Plattform von Patterns (Tim Barner Lee, 1986), Erstes Buch v. Patterns (E. Gamma, 1995).
- **Dass sich Design Patterns mit UML visualisieren lassen ist selbstverständlich und das ist in Tools auch vorhanden**
- **Wenn das Klassendiagramm so zentral ist, sollten doch viele Projekte vor uns mit ähnlichen Problemen konfrontiert gewesen sein, bis ihre Lösungen dokumentiert wurden.**

Definition

- **A Pattern is a proven Solution for a general Problem**

Motivation

- Jedes Pattern beschreibt eigentlich ein Problem, das immer wieder vorkommt und sich dadurch wiederholt auflösen lässt.
- Pattern ist das Ergebnis, als auch die Regel selbst.
- **Man muss zuerst das Problem erkennen, bevor man die Lösung einsetzen kann.**
- Komponenten selbst sind zu konkret, um z.B. zu einer vernünftigen Architektur zu gelangen.
- **Wenn eine Lösung sich bewährt hat, ist eine Katalogisierung zum Auffinden der Patterns nötig.**

Allgemeiner Nutzen

Ihre Verwendung zur Beschreibung von Problemlösungs-Beziehungen bringen folgende drei Vorteile:

- Ein gemeinsames Vokabular und Glossar
- Ihr Name reicht aus, um über diverse Alternativen zu sprechen
- **Dokumentations-, Bau- und Lernhilfe**
- **Da Patterns für allgemeine Probleme, die immer wieder vorkommen, Lösungen anbieten, liegt es nahe sie für den gesamten Lehrbereich im Fach Softwarebau, Management oder Engineering einzusetzen**
- **Erweiterung zu bestehenden Techniken in der OO-Welt**

Katalog

Muster besitzen fünf fundamentale Bestandteile:

- 1. Der Mustername benennt das Problem, die Lösung und die Konsequenzen in ein bis zwei Worten.**
- 2. Das Problem beschreibt das Umfeld des Musters. Es wird zuerst der Kern des Problems allgemein beschrieben.**
- 3. Kontext und Beschreibung der Situation, in der das Pattern sich einsetzen lässt.**
- 4. Die Lösung spezifiziert die Elemente, ihre Beziehungen**
- 5. Konsequenzen die sich aus der Anwendung ergeben**

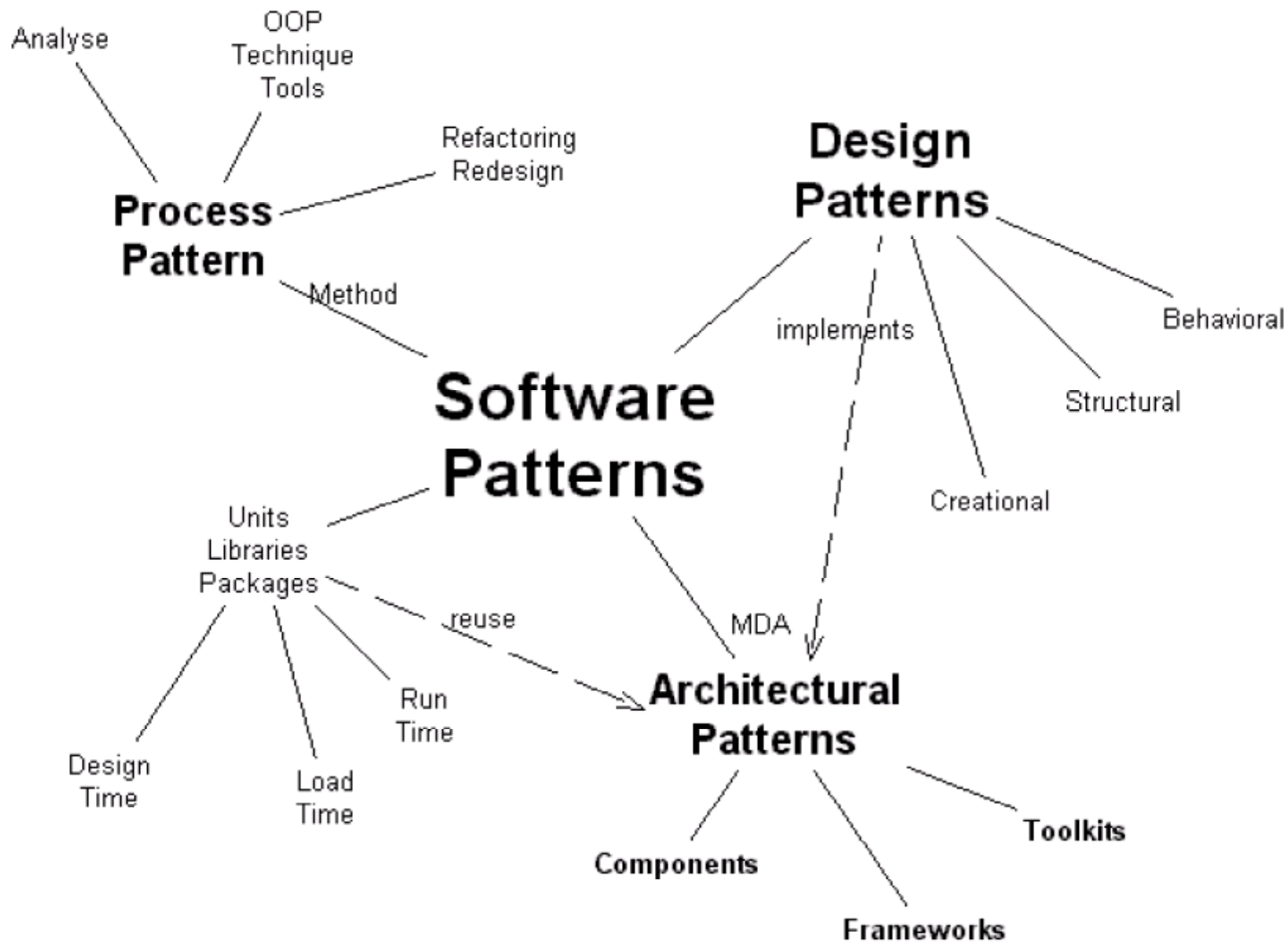
Kategorisierung



- Analyse Patterns bei Anforderungen
- Prozess Patterns bei Vorgehensmodellen
- Architektur Patterns bei der Verteilten Systemen

Design Patterns

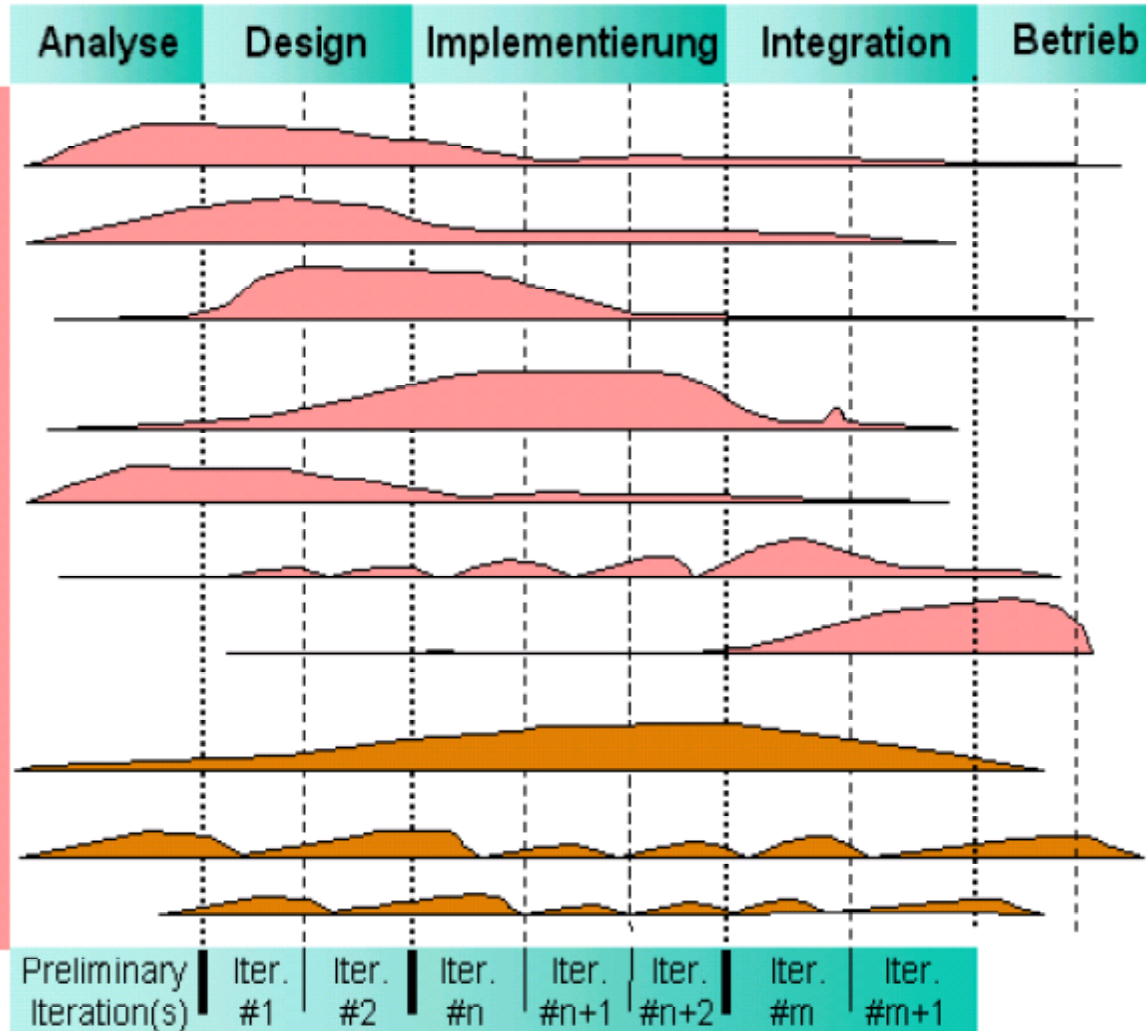
- **Creational Patterns** beschäftigen sich mit der dynamischen Erzeugung und Wiederbelebung von Objekten unabhängig der Typen
- **Structural Patterns** beschreiben den statischen Zusammenhang von Objekten und Klassen, die andere Klassen binden oder zu Strukturen führen
- **Behavioral Patterns** charakterisieren das dynamische Verhalten von Objekten und Klassen



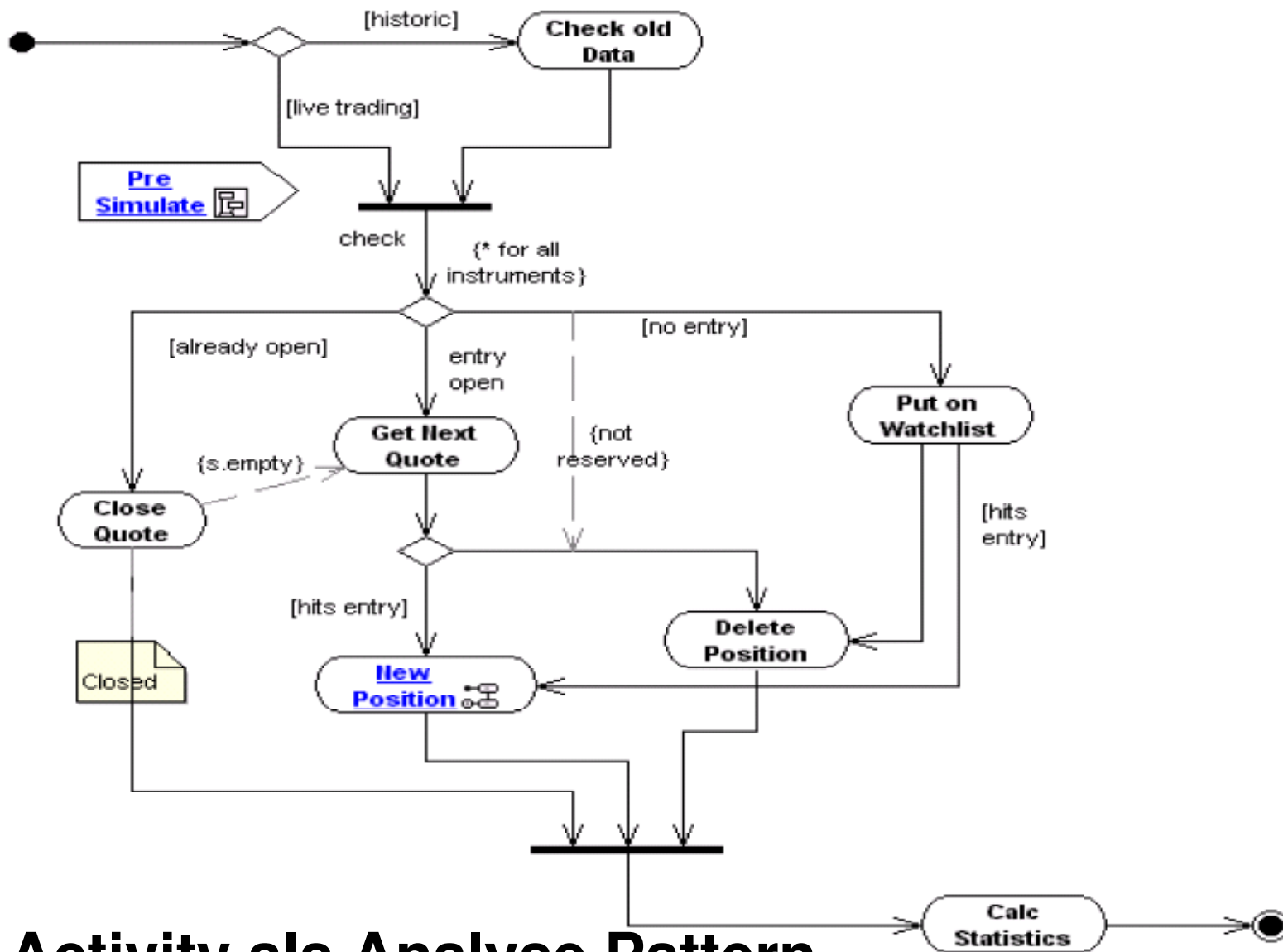
VModell

Aktivität - Produkt

Phasen



Iterationen



Activity als Analyse Pattern

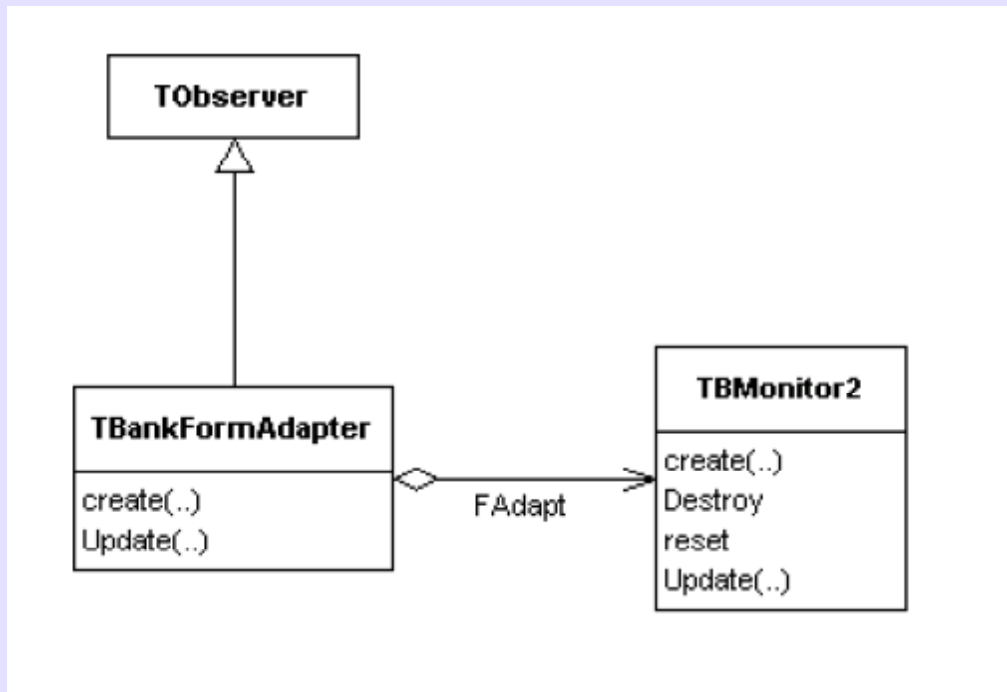
Adapter



- **Passe die Schnittstelle einer Klasse an ein andere erwartete Schnittstelle an.**
- Ein Adapter läßt Klassen zusammenarbeiten, die wegen inkompatibler Schnittstellen sonst nicht fähig wären.
- In den meisten Fällen sind Typen nicht kompatibel zueinander
- Bsp.: Ein Form klinkt sich in einen Adapter ein, d.h. es macht seinen Typ dem Adapter über den Konstruktor bekannt.
- Da der Adapter schon von TObserver abstammt, stammt nun indirekt auch das neue Form von TObserver ab

Adapter UML

- Ein inkompatibler Monitor wird typenkompatibel



Adapter Code



```
TBankFormAdapter = class(TObserver)
private
  FAdapt: TBMonitor1;
public
  constructor create(aForm: TBMonitor1);
end;
```

Die eigentliche Kopplung des Objektes an den Adapter geschieht im Konstruktor:

```
constructor TBankFormAdapter.create(aForm:TBMonitor1);
begin
  inherited Create;
  FAdapt:= aForm;
end;
```

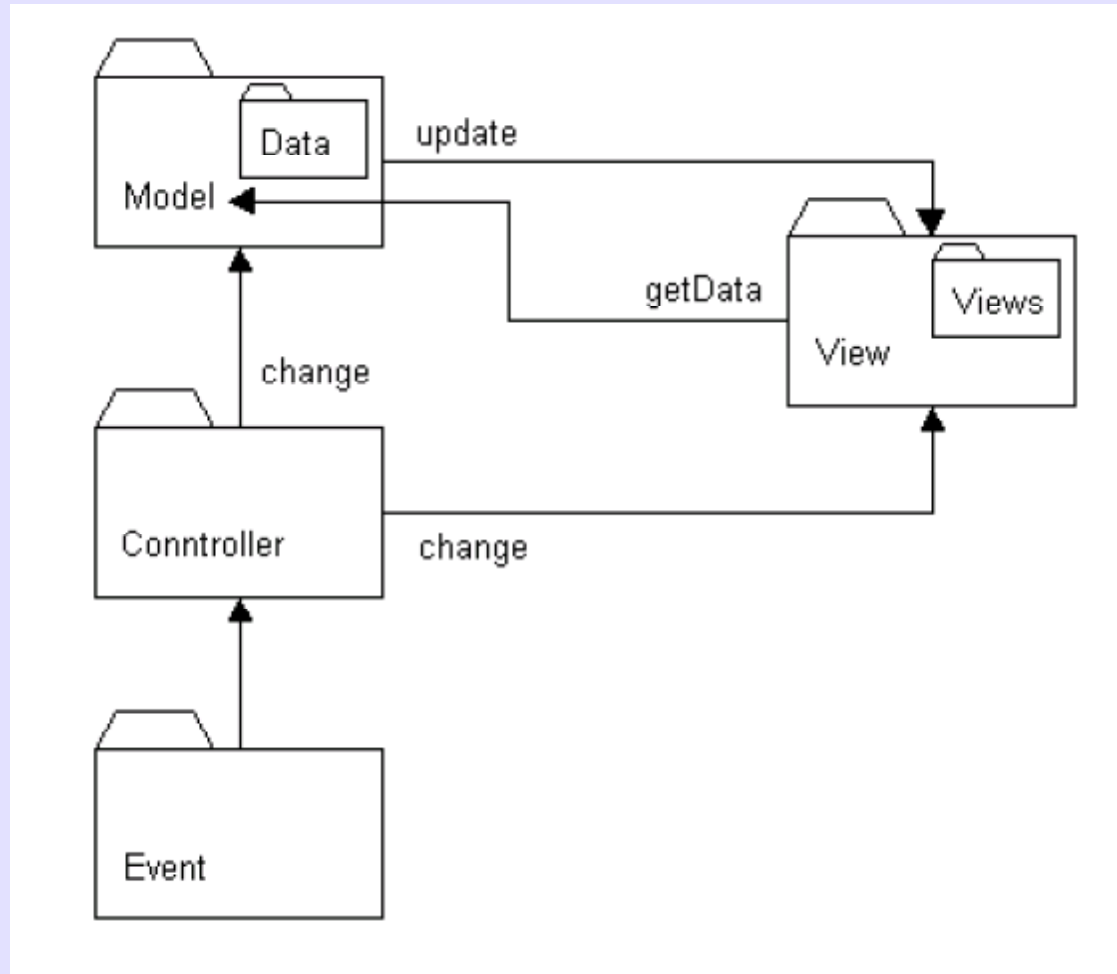
Observer



- Definiere eine Abhängigkeit von 1 : n zwischen Objekten, so dass bei einer Zustandsänderung alle registrierten Objekte benachrichtigt werden und sich dann automatisch aktualisieren können.
- **Mit den Behaviorals gilt der Observer als Star und Liebling.**
- **Mechanismen sind raffiniert und besitzen hohen Stellenwert. Im Zusammenhang mit dem MVC Prinzip, ist das Observer bestens bekannt.**
- Es braucht eine Klasse, die als Model die einzelnen Objekte die benachrichtigt werden wollen, registrieren kann

Observer UML

- Das MVC Prinzip



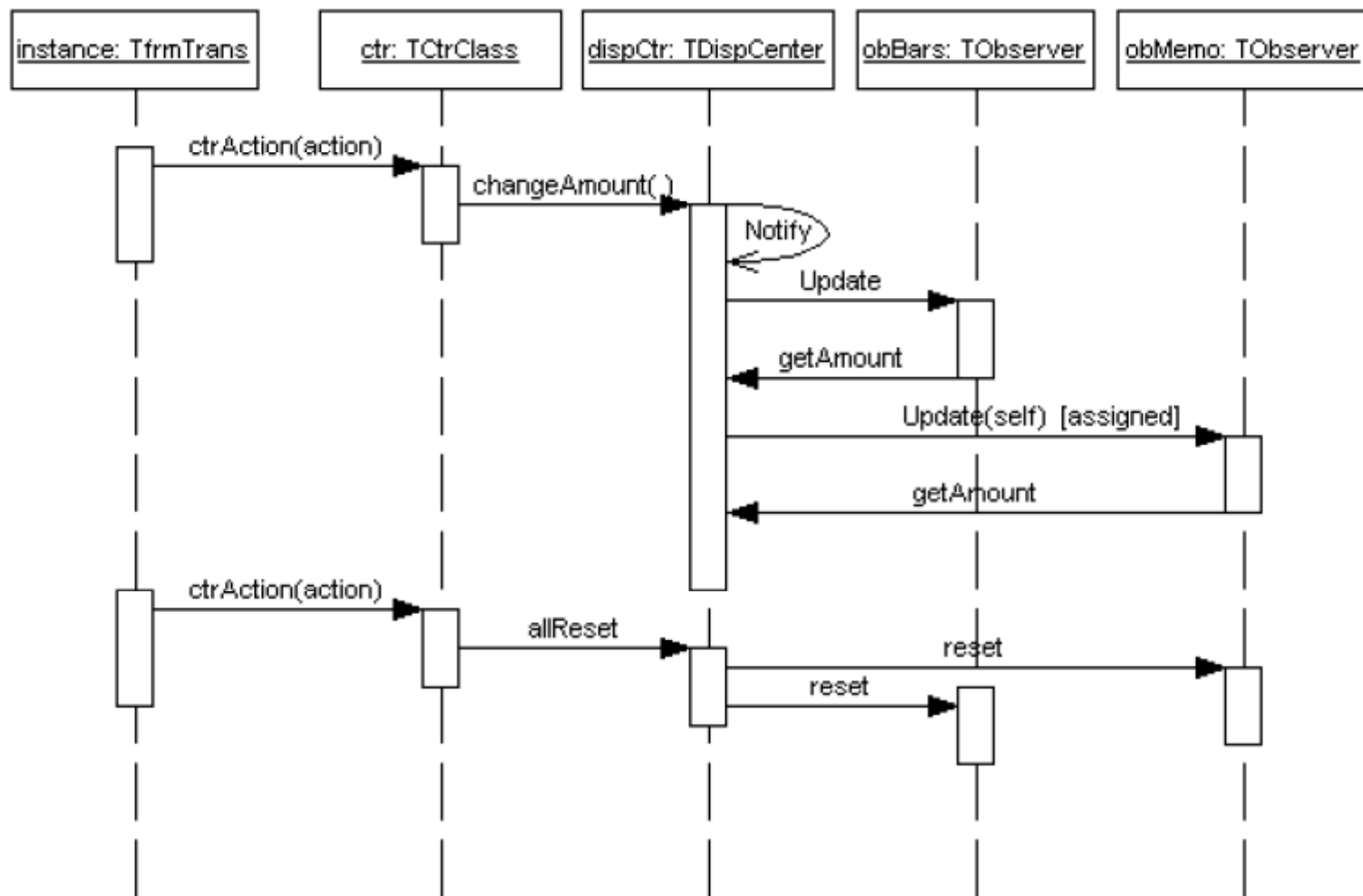
Observer Code



```
procedure TDispCenter.changeAmount(value: double);  
begin  
  if value <> 0 then begin  
    FAmount:= value;  
    Notify;
```

Die Methode Notify ist nun angewiesen, dass alle Objekte die Methode Update besitzen, damit die Nachricht ankommt:

```
procedure TDispCenter.Notify;  
begin  
  for i:= 0 to pred(FObservers.Count) do  
    TObserver(FObservers.Items[i]).Update(Self);  
end;
```

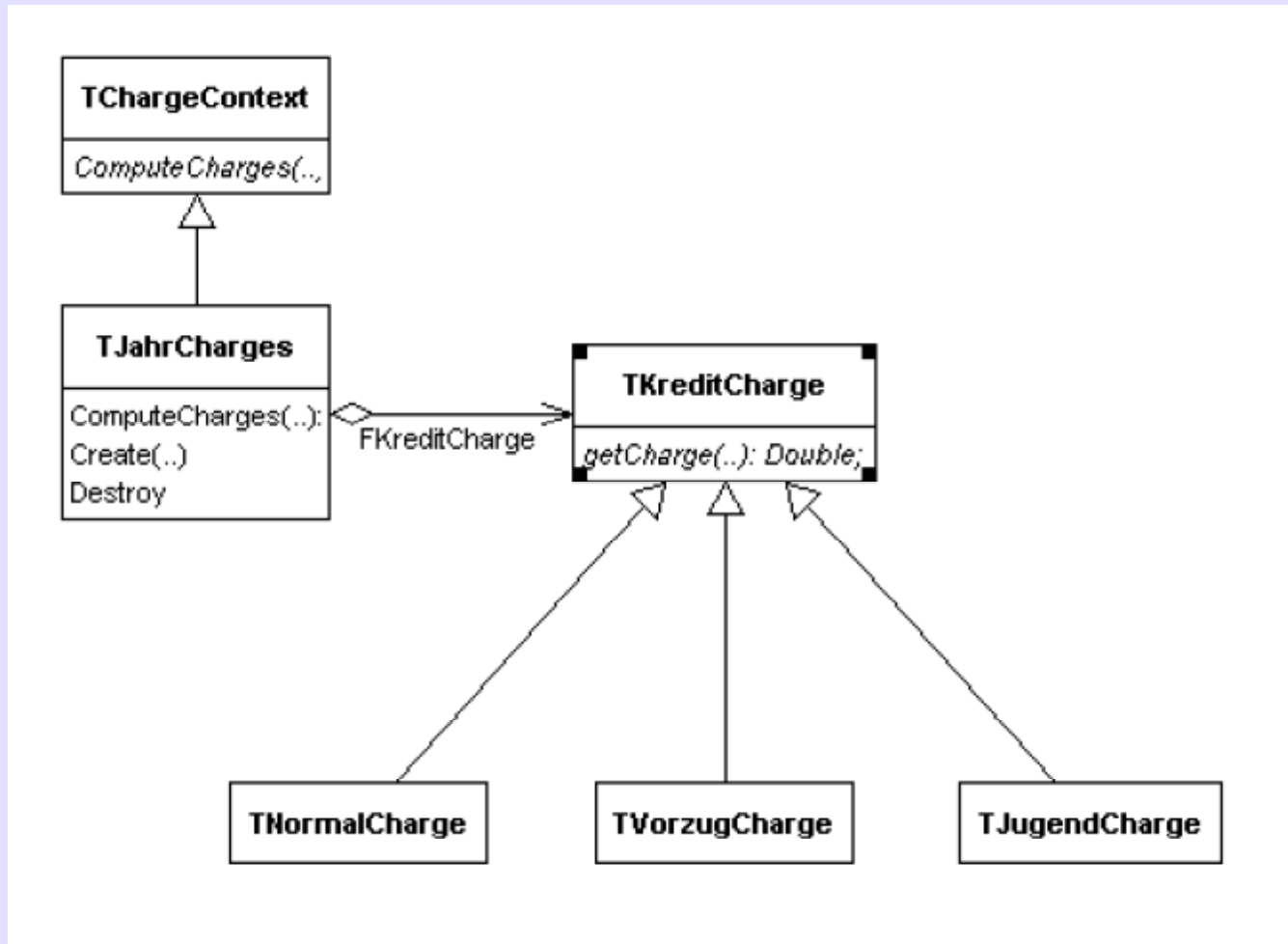


Strategy



- Bestimme eine Gruppe von Algorithmen, kapsle jeden einzelnen und ermögliche deren Austauschbarkeit während der Laufzeit. Das Strategiemuster ermöglicht den Algorithmus unabhängig vom Aufrufer zu variieren.
- **Das Strategiemuster legt den Fokus auf die Algorithmen und deren Aufgabe etwas zu berechnen. Ein Simulator, zyklische Berechnungen oder eine Rezeptur einer SPS leben von Strategien und dem zugehörigen Muster.**

Strategy UML



Strategy Code



- **Mit der Uebergabe der Referenz im Konstruktor erreichen wir maximale Flexibilität, mit oder ohne benannte Instanzen:**

```
FNormalCharges: TChargeContext;  
FPreferredCharges: TChargeContext;  
FTrialCharges: TChargeContext;
```

```
FNormalCharges:=  
  TJahrCharges.Create(TNormalCharge.Create);  
FNormalCharges.Free;
```

Master Slave



- Ermöglichte einem Master, die zu lösende Aufgabe zwischen gleichberechtigten Slaves aufzuteilen, die fehlertoleranten und genauen Ergebnisse zu einem Gesamtergebniss zusammenzuführen.
- **Das Master Slave Muster kennt Fehlertoleranz indem bei Ausfall eines Slaves der Master dem Client trotzdem ein Resultat garantieren kann.**
- **Genauigkeit wird durch den Vergleich der einzelnen Slaves erreicht. In der Industrie oder Automation gebräuchliches Muster.**

Master Slave Code



- **Mit der Uebergabe eines Hash gewinnt der erste konfliktfreie Wert durch seine Eindeutigkeit:**

```
function MakeHash(const s: string): Longint;  
{small hash maker}  
var  
  I: Integer;  
begin  
  Result:= 0;  
  for I:= 1 to Length(s) do  
    Result:= ((Result shl 7) or (Result shr 25)) + Ord(s[I]);  
end;
```

Praxisbezug

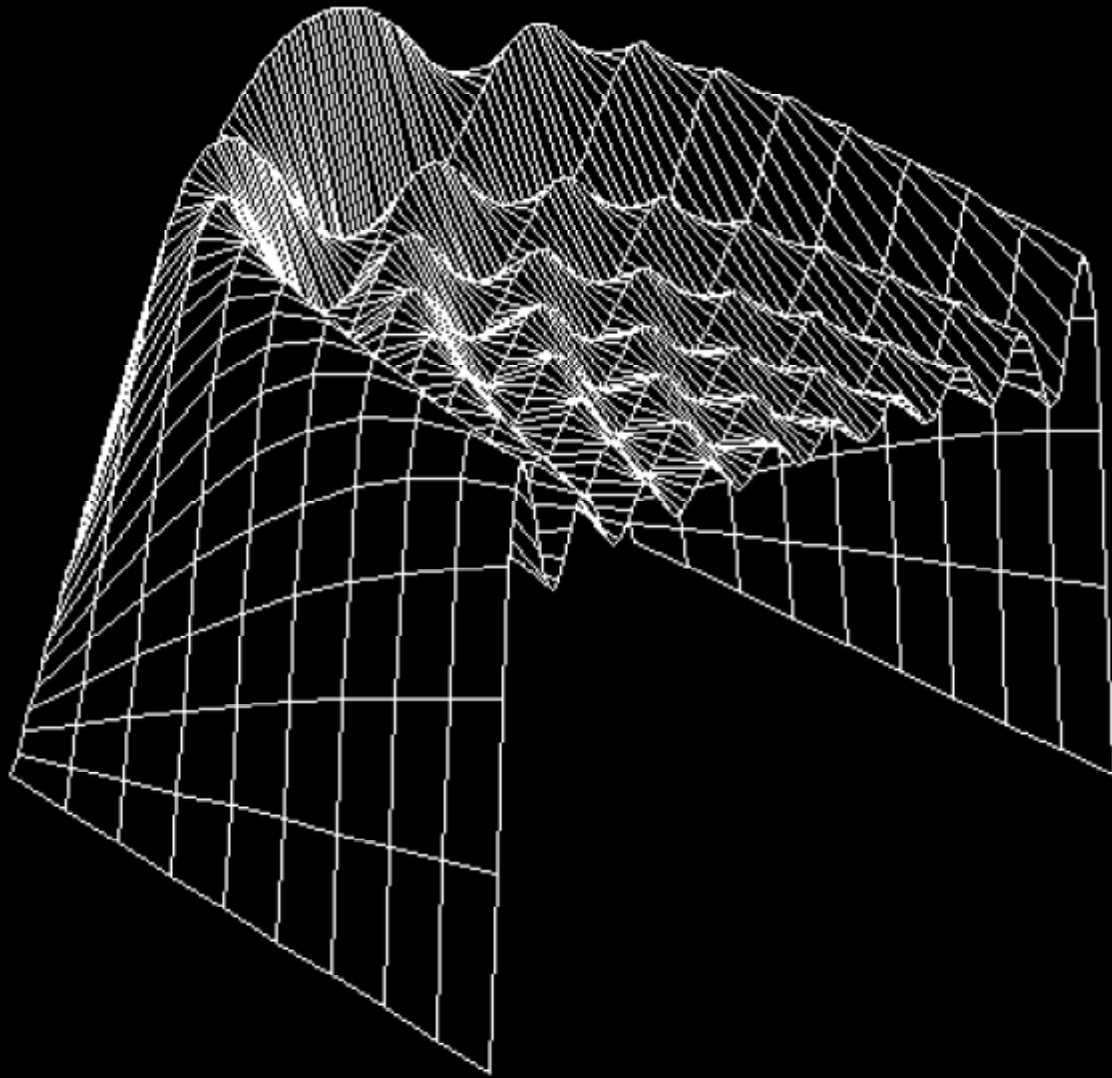


- Ermöglichte in allen Anforderungen und Geschäftsprozessen eine einheitliche Notation.
- **Folien: Prozess IT-Systeme der armasuisse und Planungshandbuch GST**
- **Patterns lassen sich in Packages verwalten**
- **MDA umfasst das Konzept der Patterns bei Transformationen**
- **Generierter Code 60-70% heisst nicht gesparte Zeit!**
- **XM als Extreme Modelling einführen
(Ausführbare Spezifikation, Simulation)**

Fazit

- Patterns als Tech. Anf. festlegen
- Patterngenerator in Tools prüfen
- Architektur vor Klassen
- Design ist Implementierung der Architektur
- Module vor Komponenten
- Fragen Sie nach DP inside bei Sourcen
- Koppeln Sie DP an die Dokumentation
- Code Patterns versus Design Patterns
- MDA in UML 2.0 baut auf Patterns
- Teile und herrsche
- Auch einer der aus dem Rahmen fällt kann noch im Bilde sein ...
- Fragen: max@kleiner.com





Viel Spass mit den täglichen Mustern