

Das ist neu in UML 2.0

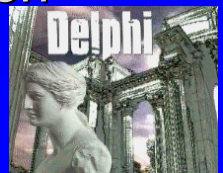
max.kleiner.com

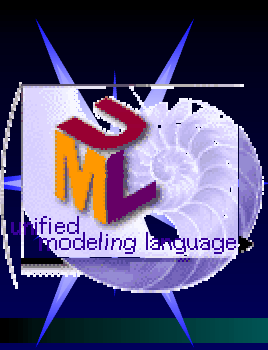




Die 2.0 ist formal in 3 Gebiete aufgeteilt:

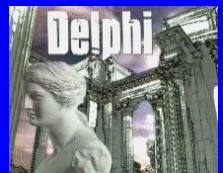
- **UML Infrastructure**, Vereinfachung und Modularisierung des Metamodells, Profiles und Stereotypen. Das Metamodell ist frei von sprachabhängigen Konstrukten (wie C++ oder ADA) und sprachneutral, d.h. in OCL definiert.
- **UML Superstructure**, mit den eigentlichen Erweiterungen bezüglich der Notation, Syntax und der Elemente, somit für den Benutzer sichtbar und verwertbar.
- **UML OCL /Interchange**, Integration in das Metamodell, so daß die Spezifikation Modell -> Code eine formale Sprache beinhaltet. Die OCL ist zu einer generellen Ausdruckssprache erweitert. Besserer Modellaustausch durch XMI (XML Metadata Interchange) z.B. mit Layout Informationen zwischen den Tools.

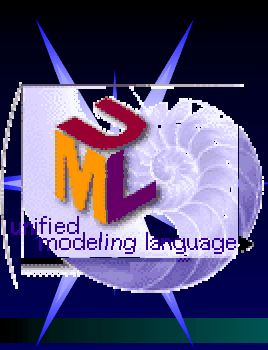




Open or programming for change

- ☯ UML ist eine Sprache zur Beschreibung von Softwaresystemen, die eine Vereinheitlichung der unterschiedlichen Ansätze zur OO-Modellierung darstellt und eine MDA Generierung erlaubt.
- ☯ UML stellt auch einen Prozess <V> zur Verfügung, der entlang der Entwicklung Diagramme in den entsprechenden Phasen vorgibt.
- ☯ Die GUI- und Datenmodellierung als begleitende Entwicklung zur UML
- ☯ Dokumentation, Testphasen und Reviews dienen der Qualitätssicherung

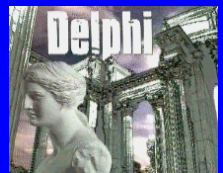




Summary of session

- ☯ Die veränderten und neuen Diagramme, ihre Notation und Semantik

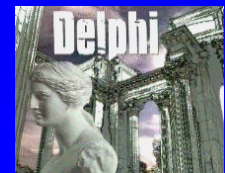
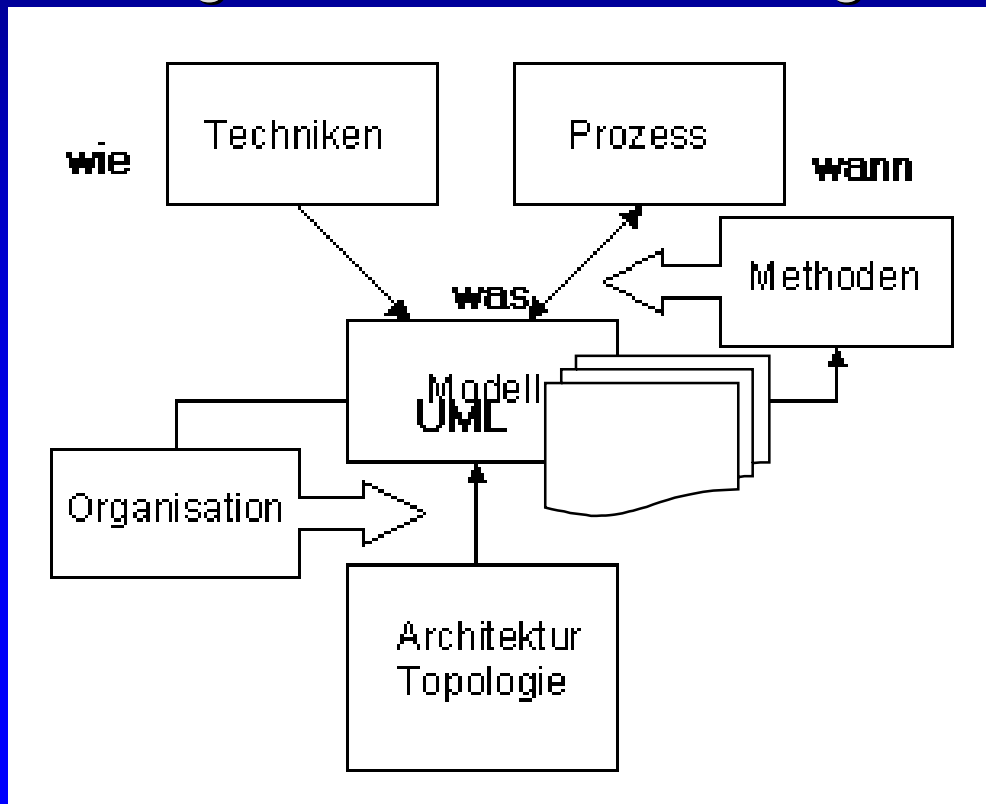
- ☯ Die Anpassung der UML 2.0 an bestimmte Domänen (Profile) mittels der MOF (Meta Object Facility) und XMI, zeigt die zusätzlichen Möglichkeiten der MDA.





Grobneuheiten

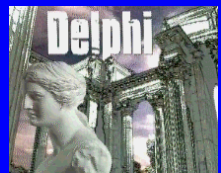
- ☯ In der Dynamikmodellierung (Zeitdiagramm, Zustandsautomat, Sequence und Activity) hat sich in den Verhaltensdiagrammen am meisten getan.





Detailneuheiten

- ☯ Für das Metamodell entsteht ein Sprachkernel
- ☯ Zusätzliche Notationen bauen auf dem Kernel auf
- ☯ Support für den Einsatz von Komponenten untereinander
- ☯ Ausbau der Geschäftsprozessmodellierung (BPM)
- ☯ Interne Struktur für Bezeichner, Schnittstellen, Klassen, Typen und Rollen
- ☯ OCL Integration als generelle Bedingungssprache
- ☯ State Event Generalisierungen und Echtzeiterweiterungen
- ☯ Erweiterungen für kompaktere und wiederverwendbare Sequenzdiagramme
- ☯ Architektur getriebene Modellierung (MDA)





Ziel: ein genaues Modell ?!

CNN, 25.09.2001



FUTURES
S&P ▲ 0.50

WAR AGAINST TERROR
PUTIN IN GERMANY FOR
ANTI-TERROR TALKS



Was ist hier falsch?



UML 2.0 Diagramme

☉ Use Case

- Initialisierung, Akteure
- Anwendungsfälle im Kontext
- allgemeine Diskussionsgrundlage

☉ Activity

- Geschäftsprozesse
- Parallele Prozesse, Signale
- Workflow...

☉ Class Diagram /Object Diagram

- Strukturierung
- OOP-Entitäten
- Relationen
- so gut wie überall...

☉ State Event /Protocol Automat

- dyn. Verhalten innerhalb Klassen
- Objektlebenszyklus
- Zustandsübergänge

☉ Deployment

- Zusammenspiel Komponenten
- Verteilungsaspekte
- Horizontale Systemarchitektur
- Netze, Protokolle und Einsatz

☉ Component /Subsystem /Composition

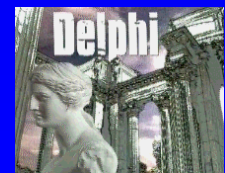
- Autonome Schnittstellen
- ausführbare Architektur Gruppen
- source, binary, executable

Packages /Collaboration Patterns

- Auswirkung v. Änderungen
- Vertikale Architektur
- Package =Unit als Modul
- Libraries, Patterns

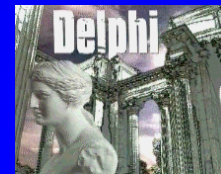
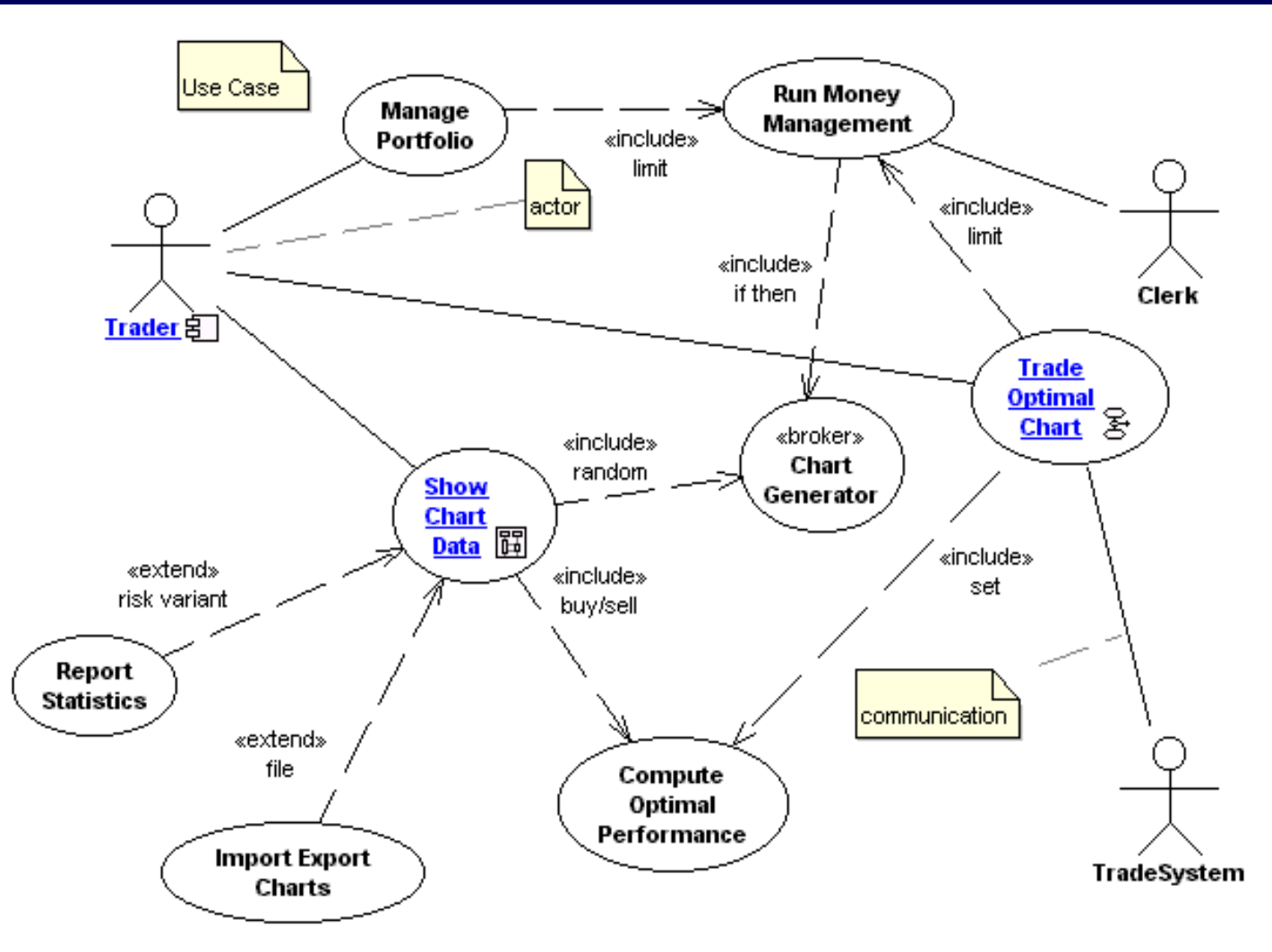
☉ Sequence Diagram /Communication /Interaction Overview /Time Diagram

- Nachrichtenfluss der Objekte
- Zeitliche Aufrufstruktur
- Dynamische Aufrufkaskaden





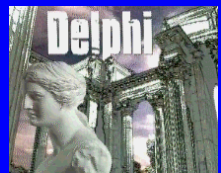
Use Case





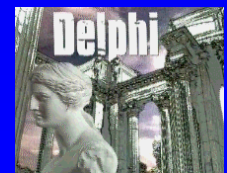
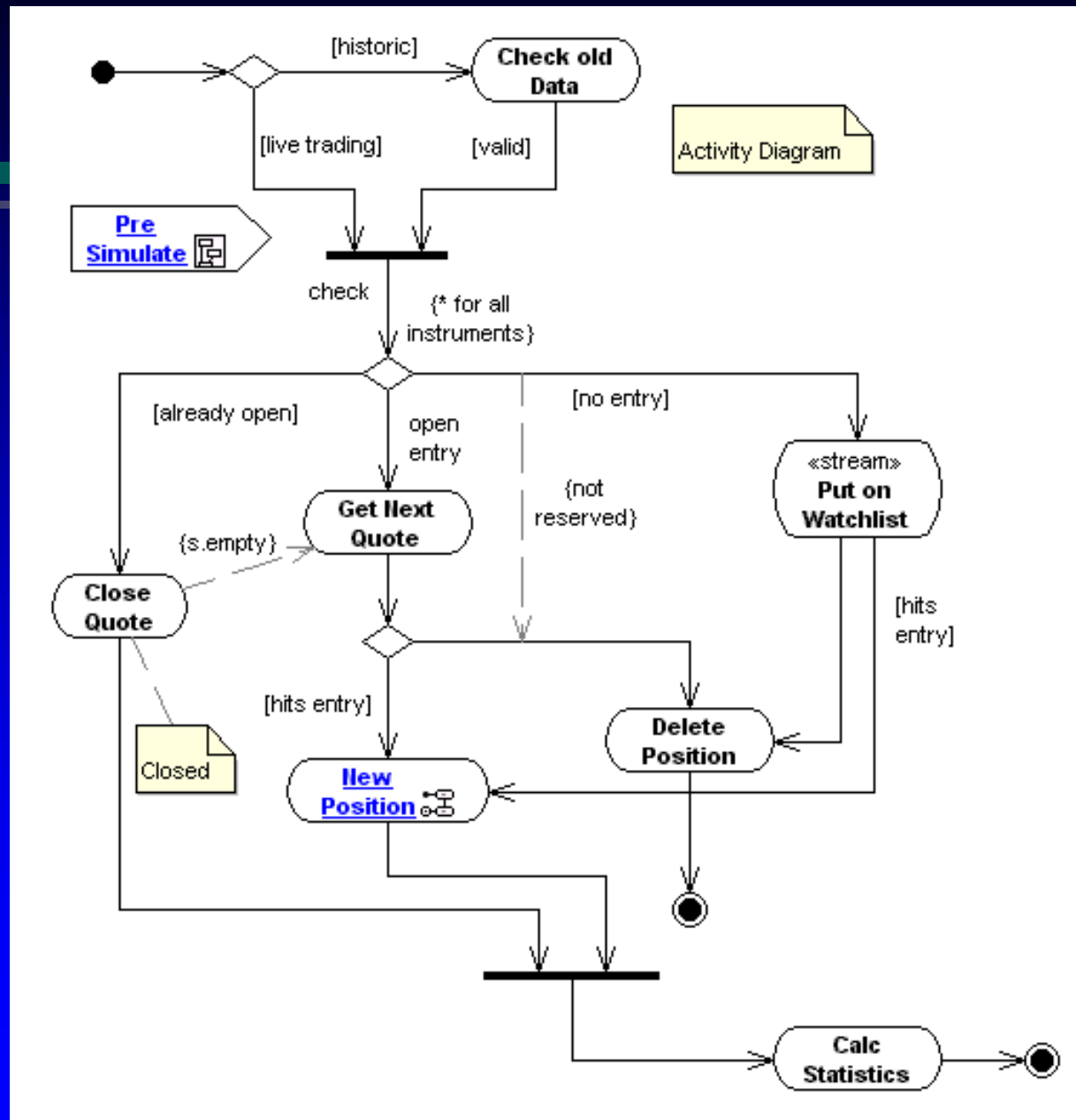
Use Case Notation (kaum Neues)

- ☯ Unter <use case> (UC) wird eine typische Handlung verstanden, die ein <actor> mit dem System ausführt.
- ☯ Ein <actor> (kann auch ein Subsystem sein) kommuniziert gegenseitig mit einem UC
- ☯ <include> wird verwendet, wenn ein Verhalten von zwei oder mehr UC gebraucht werden kann.
- ☯ <extends> erweitert einen UC mit alternativem Verhalten, welches im Gegensatz zu <include> entbehrlich ist. Achten Sie auf die Pfeilrichtung.







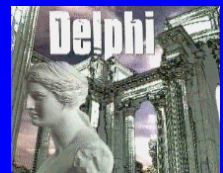
Activity





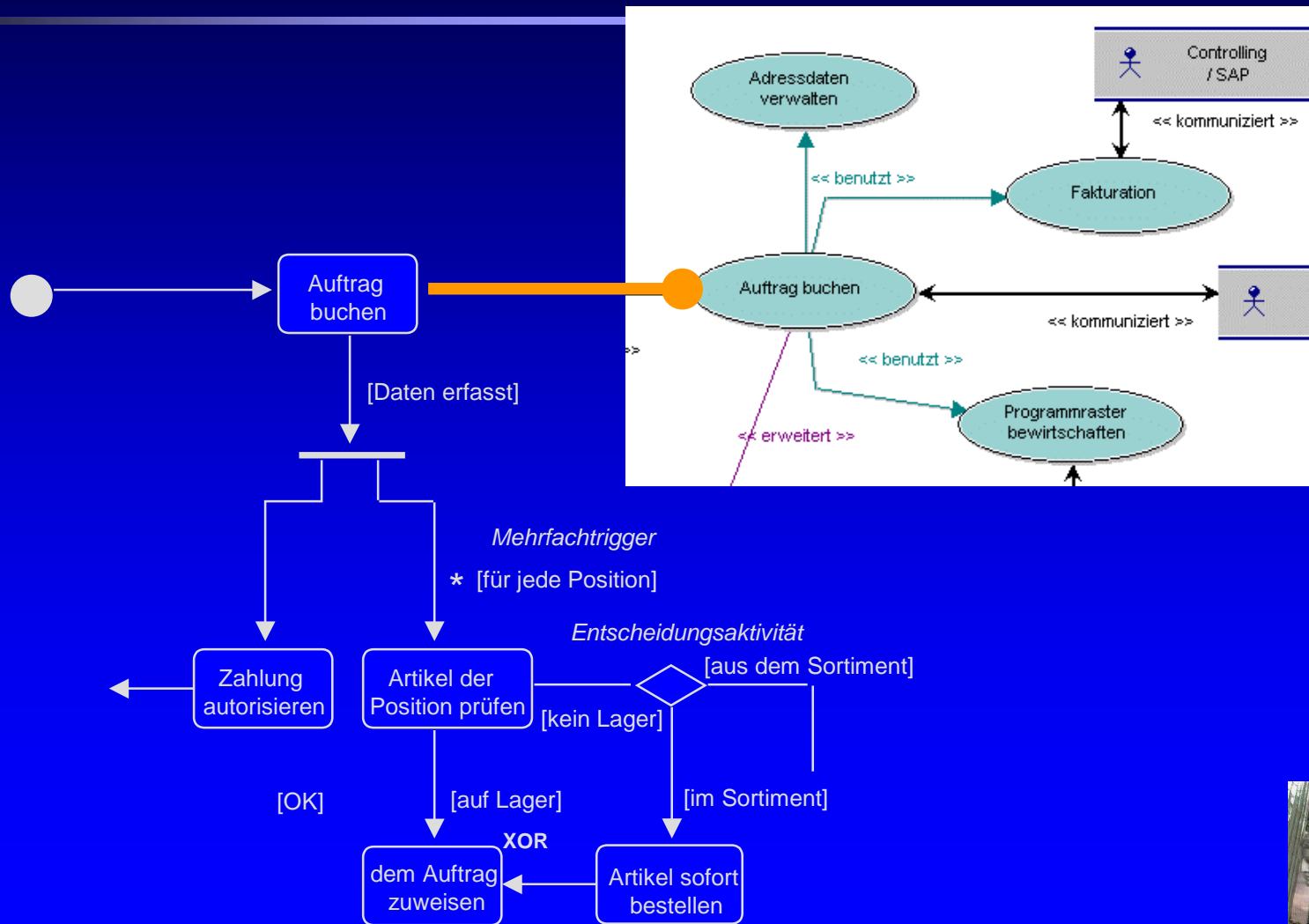
AD Notation (Neudefinition)

- ☯ AD sind nun losgelöst von den <state events> und zeigen die Semantik von Petrinetzen, geeignet für Workflow und parallele Prozesse.
- ☯ <actions> sind Zustände, in denen aufrufbare Vorgänge ablaufen <invocation> inkl. <exceptions>.
- ☯ <tokens> erfolgen zwischen den Aktionen und lassen sich mit [constraints] und [pins] näher definieren.
- ☯ <synchronisation bars> signalisieren, dass alle tokens vorhanden sein müssen, neu: Ablaufende 
- ☯ Objektknoten  sind als Parameter zu verstehen.
- ☯ Nebenläufigkeit und Mengenverarbeitung möglich.



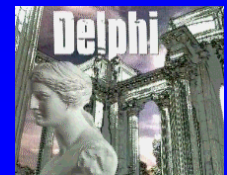
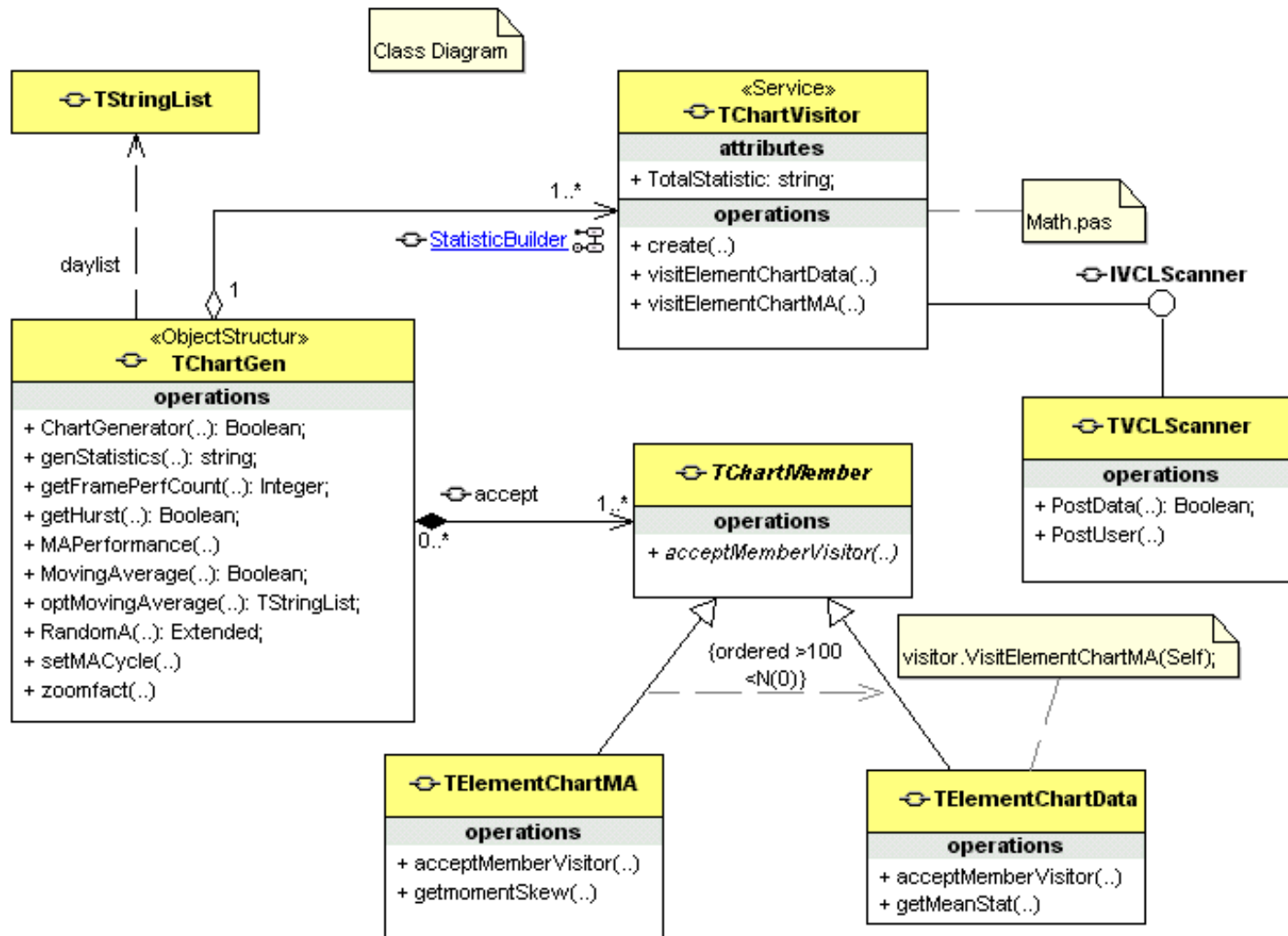


Relation Use Case - Activity





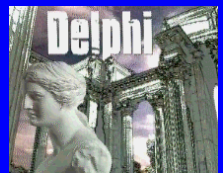
Class Diagram (wenig Neues)





Klassen Notation (neue Beziehungen)

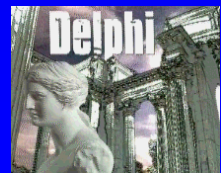
- ☯ Navigationsrichtung ist nun definierbar
- ☯ <association> ist eine Beziehung zur Laufzeit mit loser Kopplung zwischen den Objekten
- ☯ <aggregation> besitzt die Klasse ein Objekt (enge Kopplung) zur Designzeit als Attribut
- ☯ Klassen und Interfaces erhalten neu einen <part>, das ist ein Teil oder Typ einer Klasse.
- ☯ <visibility> ist der Zugriffsschutz unter Klassen:
 - ◀ private, protected, public und published wirken auf Felder, Methoden (<attributs>) und Eigenschaften





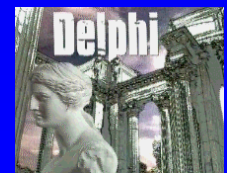
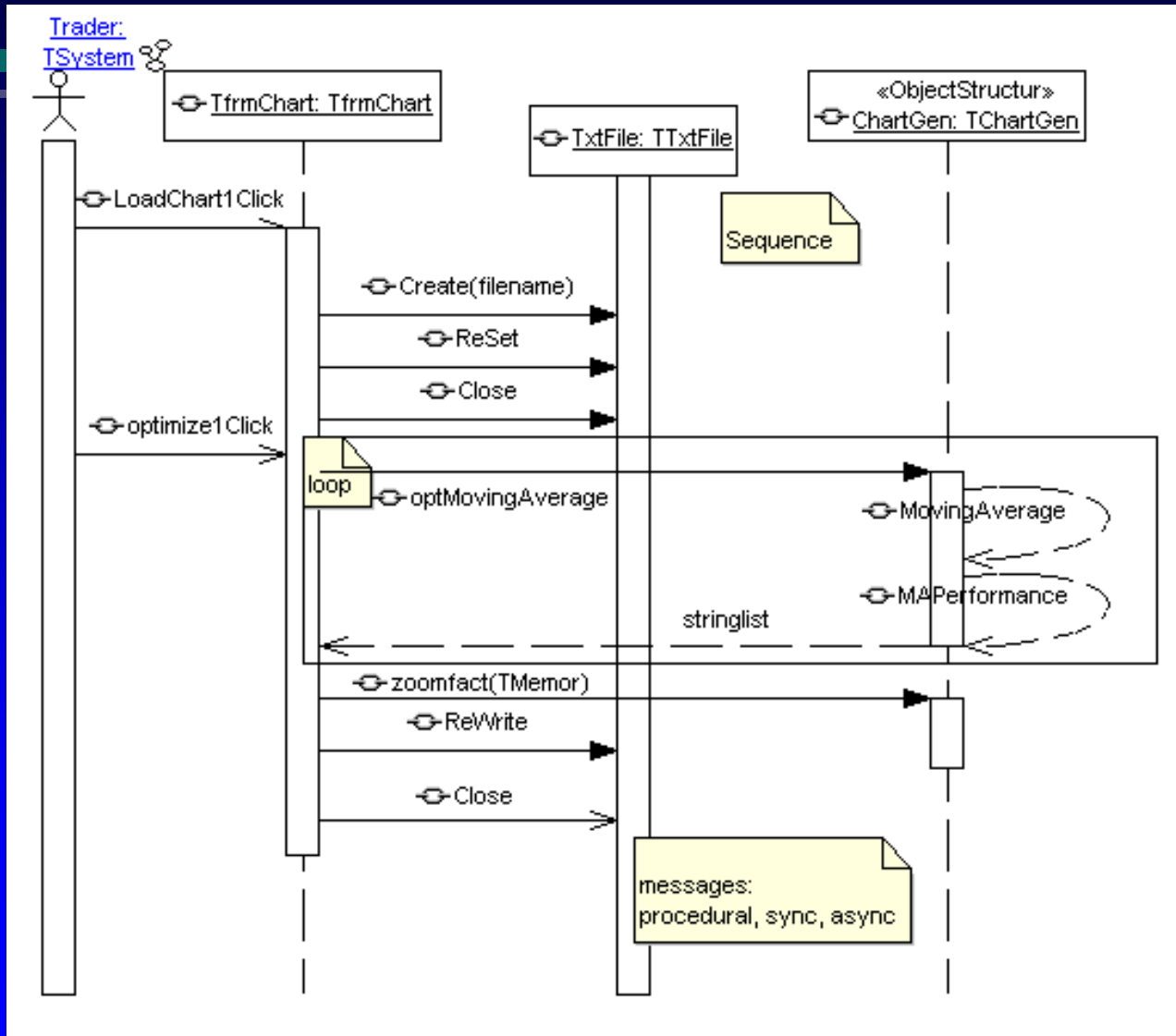
Klassenbau Neu mit Robustness

- ☯ Document (Model), Controller, View (DCV) ist ein Prinzip zur Verteilung der Funktionalität auf Klassen:
- ☯ Verarbeitungsdominante Klassen realisieren Kontrolle und Steuerung von 1 oder n Use Case und werden auf <entity> oder <control> -Klassen verteilt
- ☯ Dialogdominante Klassen sollten wenig «Wissen» enthalten und sind in <boundary> -Klassen zu finden
- ☯ Zur Klassenfindung hier drei Techniken:
 - ⚡ Anwenden von «design patterns» spart viel Zeit
 - ⚡ Textanalyse des Use Case
 - ⚡ Teamarbeit mit CRC-Karten





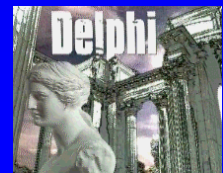
Sequence Diagram





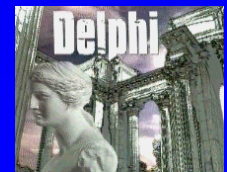
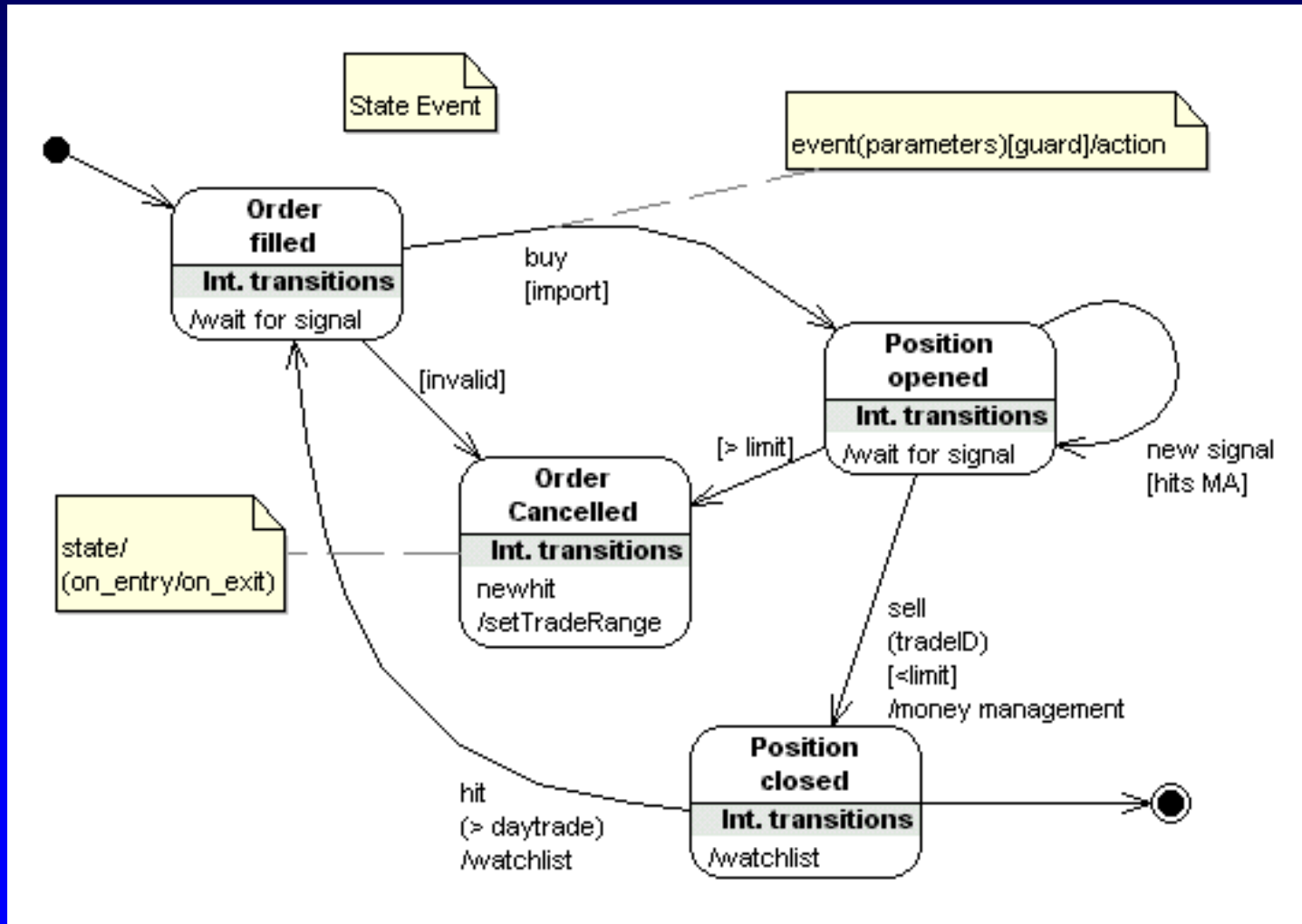
Sequence Notation (Erweitert)

- ☉ Neu aufgenommen wurde das Timing Diagramm aus der Elektrotechnik und Echtzeitverarbeitung.
- ☉ <gates> Operatoren erlauben <alt>(Verzweigung), <loop>(Schleife), <break>(Ende), <opt>(Optional), <par>(Threads), <ref>(Verweis) → Ein- Ausstieg
- ☉ Jede <message> wird durch einen Pfeil zwischen dem <focus of control> zweier Objekte dargestellt. Mit n:Objekten lassen sich <scenarios> modellieren.
- ☉ Kollaboration heisst neu Kommunikationsdiagramm:
 - synchroner, asynchroner Aufruf, global, local, parameter, self
 - Iterationen <*> und Bedingungen []





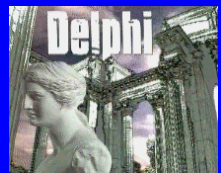
State Event einer Klasse





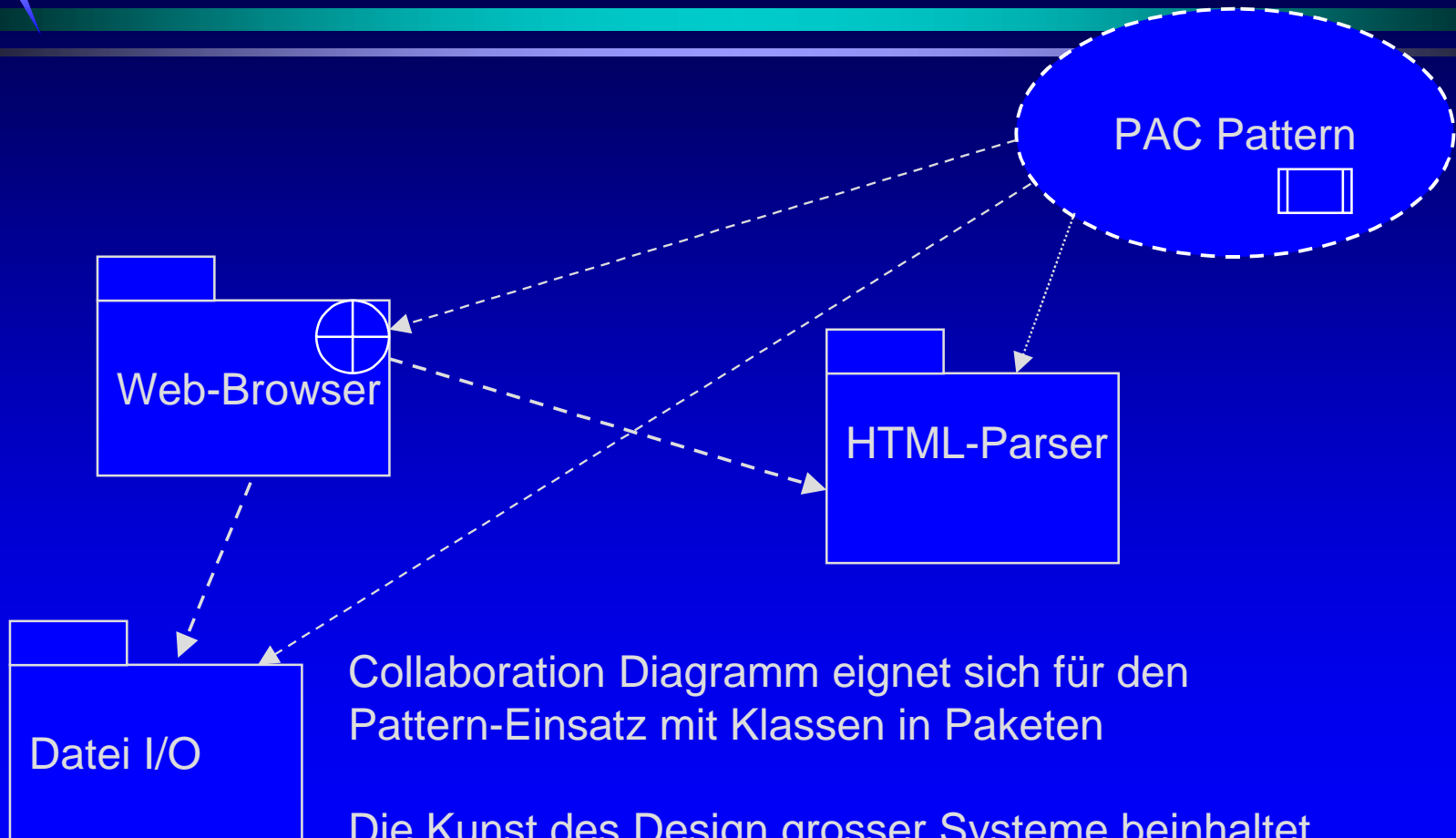
State Event Notation (Erweitert)

- ☯ Aus den IAD lassen sich Zustandsdiagramme mit `<scenarios>` pro Objekt vollständig entwickeln, jedoch nur «interessante» Klassen lohnen sich
- ☯ Die Syntax einer `<transition>` zwischen den Zuständen ist: `<event(argument) [condition] /operation(>`
- ☯ Ein `<event>` kann durch eine Bedingung `<guard>` oder eine Nachricht ausgelöst werden, z.B «prüfe Zahlung»
- ☯ Neu sind Signale zum Senden und Empfangen...
- ☯ Neu: Protokollautomat determ. Operationsfolge
- ☯ Tip: Ergebnis als Zustand (partizip) modellieren





Packages (neu mit Patterns)



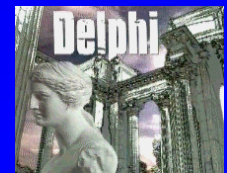
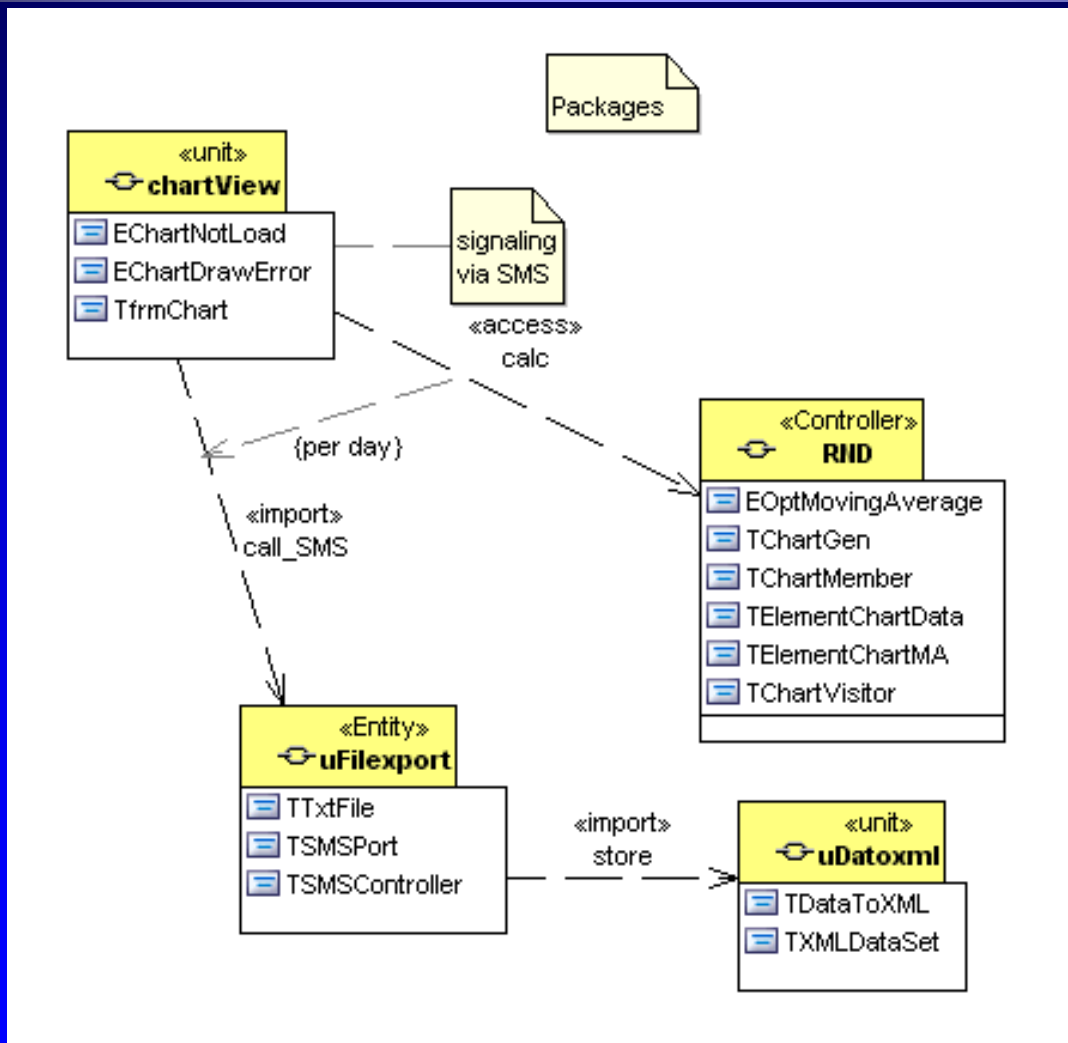
Collaboration Diagramm eignet sich für den Pattern-Einsatz mit Klassen in Paketen

Die Kunst des Design grosser Systeme beinhaltet die Minimierung v. Abhängigkeiten, welche die Auswirkungen v. Änderungen minimiert





Packages

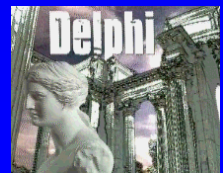




Packages Notation (kaum Neues)

- ☯ Ein `<package>` ist eine nach fachlich oder techn. Themen organisierte «Übersetzungseinheit» von Klassen mit möglicher Verschachtelung.
- ☯ `<dependencies>` zeigen an, dass bei einer Änderung des `<package>` an der Pfeilspitze das `<package>` am anderen Ende der gestrichelten Linie ev. geändert oder neu übersetzt werden muss

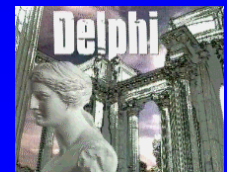
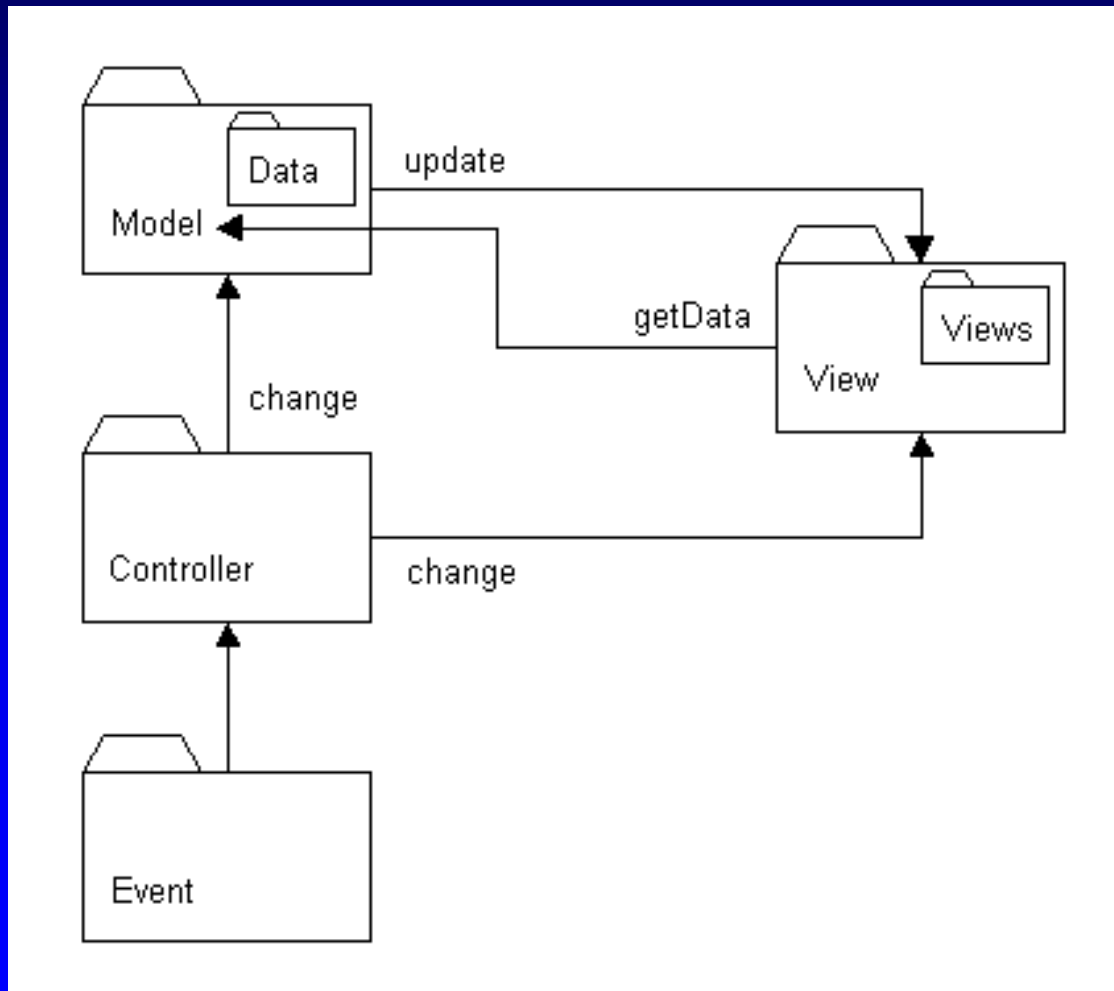
Neu läßt sich nun zwischen `<access>` und `<import>` unterscheiden, bei `<access>` greift eine Klasse auf die öffentlichen Elemente der anderen Klasse zu, bei `<import>` wird der ganze Geltungsbereich (Namespace) eines Paketes dem anderen Paket hinzugefügt oder eben importiert.





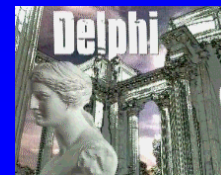
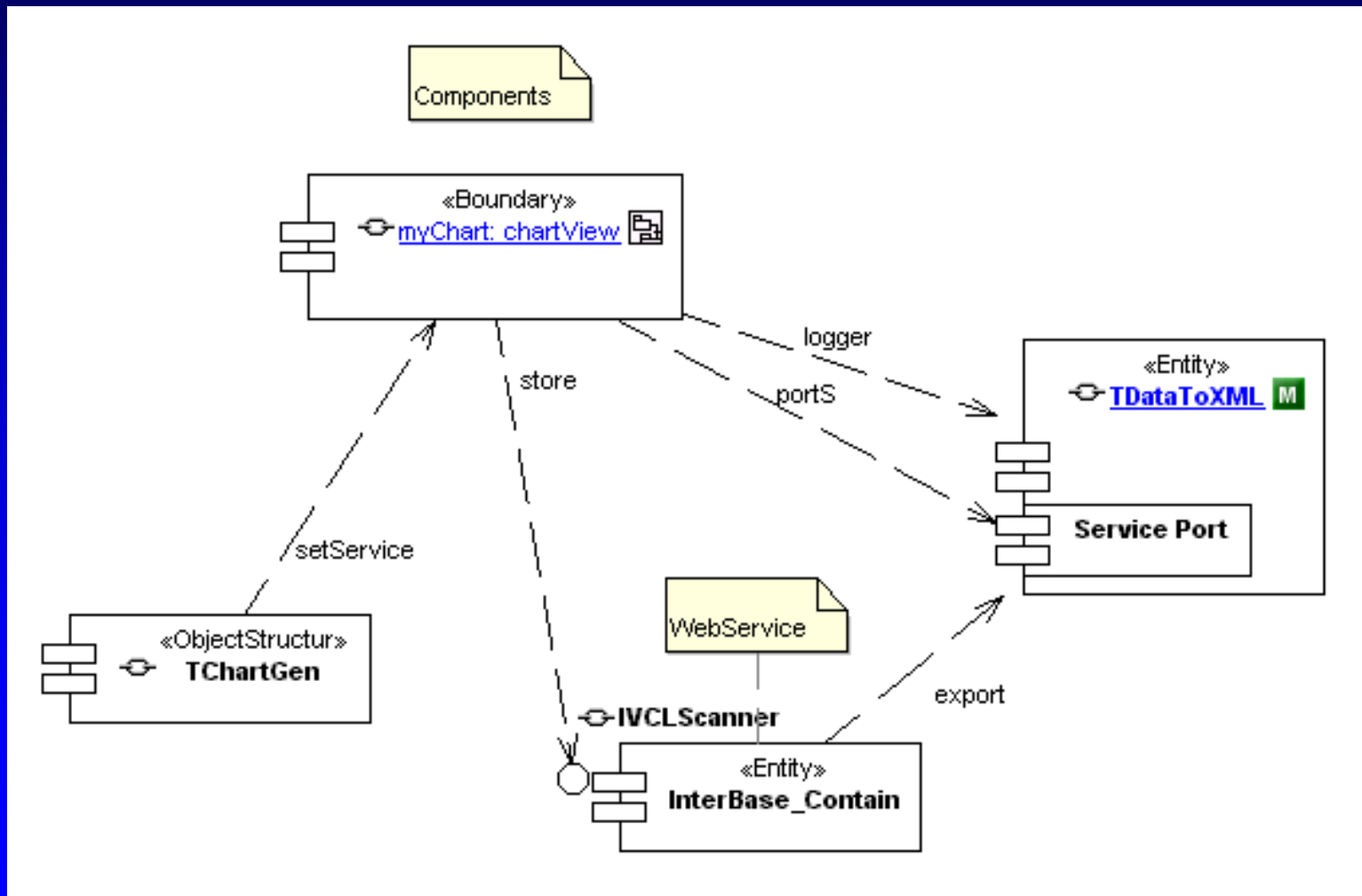
SubSystem-Diagramm (Neu)

Architektonische Zusammenhänge von Komponenten



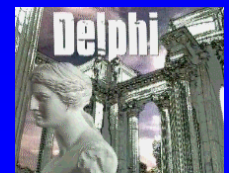
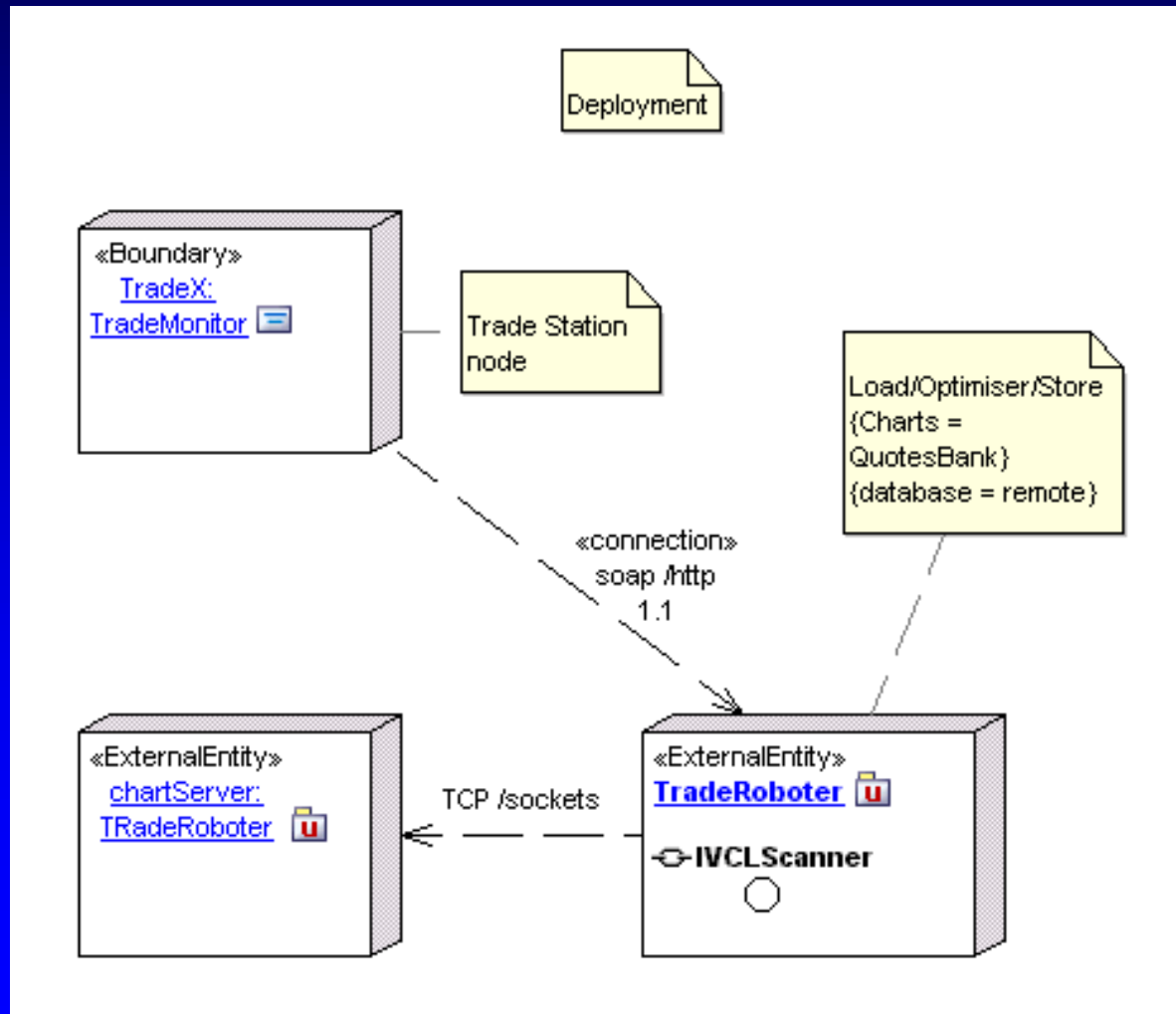


Component





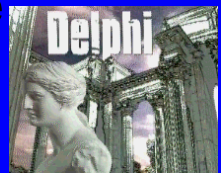
Deployment





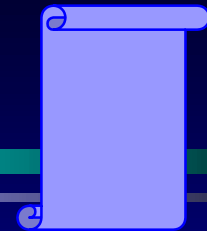
Deployment Notation (eher Neu)

- ☉ Ein Verteilungsdiagramm (DEP) zeigt die realen Beziehungen zwischen Soft- und Hardwarekomponenten im integrierten und installierten System
- ☉ `<components>` Neu ermöglichen `<ports>` und `<connectors>` zwischen einer angebotenen (Technik, Typ, Laufzeit und Signatur) und einer erforderlichen Schnittstelle das mögliche „Zusammenstecken“ prüfen zu lassen um z.B. festzustellen, daß eine MSCOM nicht zu einem EJB oder einer CLX kompatibel ist. Somit zeigt das DEP, in welchem `<node>` jedes `<package>` oder `<component>` im System abläuft.
- ☉ Das Deployment ist mit den `<deployment descriptors>` erweitert, so daß ein informeller Text in absehbarer Zeit auch ein Tool generieren kann und als Manifest-Datei in den Knoten einpackt. Diese Angaben lassen sich dann auf der Zielplattform in eine Konfigurationsdatei umwandeln (MDA).





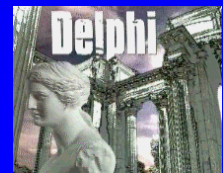
4 Schritte der MDA

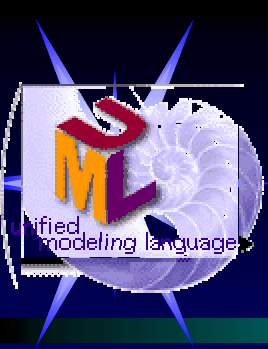


- ☹ 31% der Client/Server Projekte werden vor Vollendung gestrichen oder nicht eingesetzt
- ☹ 57% der Projekte überschreiten die Kosten mit ca. 40%

-
- ☹ UML Profile (zB. DataSetProvider)
 - ☹ PIM (Klassen Lohnverwaltung, fachlich)
 - ☹ PSM (CLX Library, technisch)
 - ☹ Code (Deployment Descriptor Components)
-

Jede Sprache mit der zugehörigen Entwicklungsumgebung ist von einer inneren Architektur von Subsystemen abhängig, z.B. dem Framework der GUI, der Struktur zwischen Interface und Implementierung oder den typischen Datenbankzugriff-Komponenten mit ihren Connection-Strings. Dies sollte standardisiert generiert werden!





OCL konkret

z.B.: Klasse

TKinoBelegung:

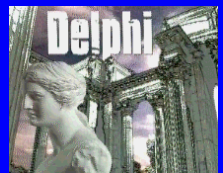
```
age >= 12 or parent.age > 16
```

```
context Salesperson inv:
```

```
clients->size() <= 100 and clients->
```

```
forAll(c: Customer | c.age >= 40)
```

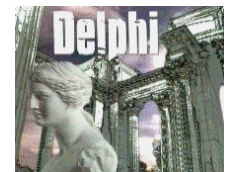
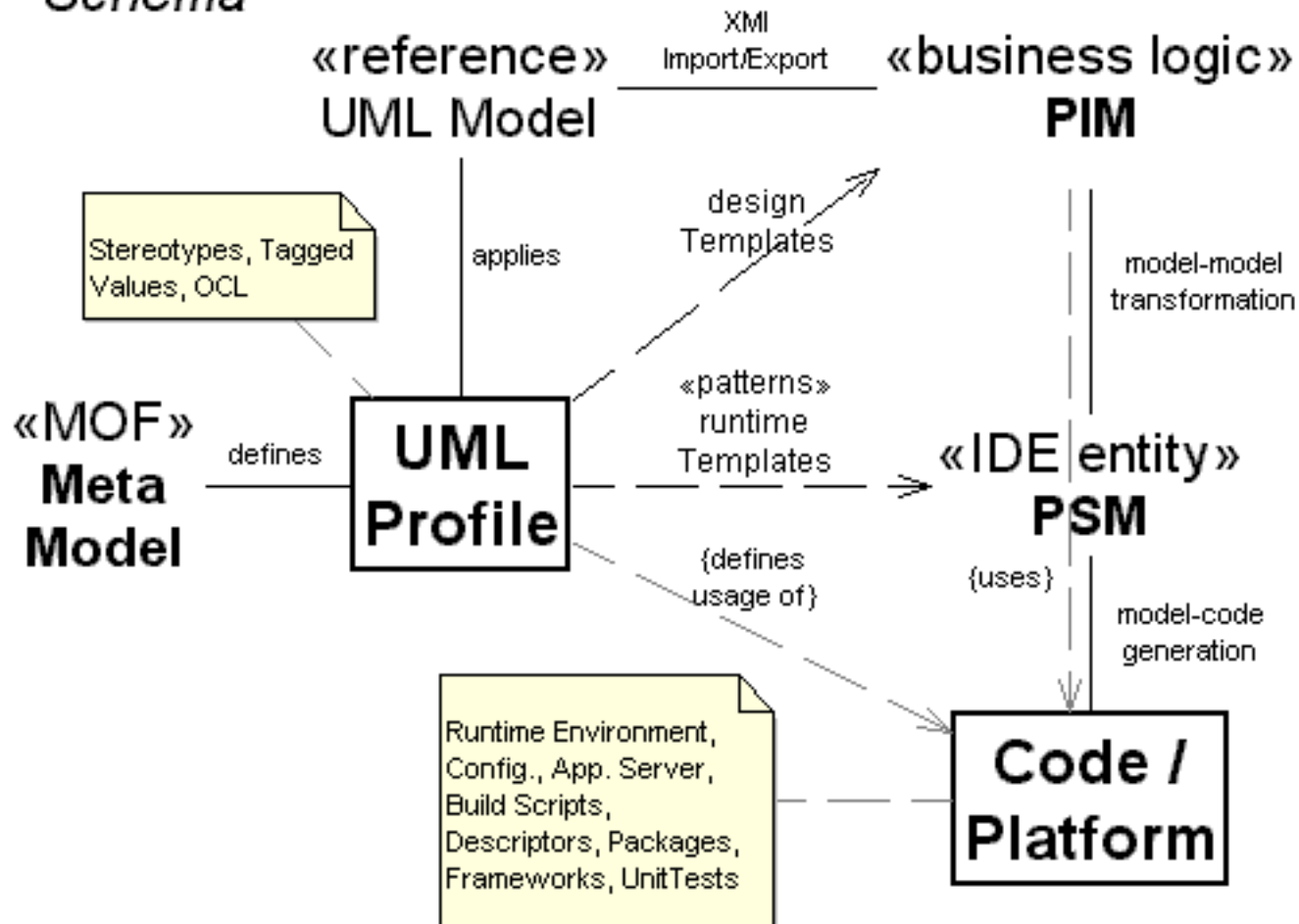
Die OCL ist eine reine Bedingungssprache, will heißen, jede Auswertung hat niemals eine Änderung der Daten oder des Modells zur Folge.





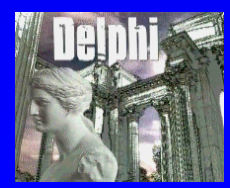
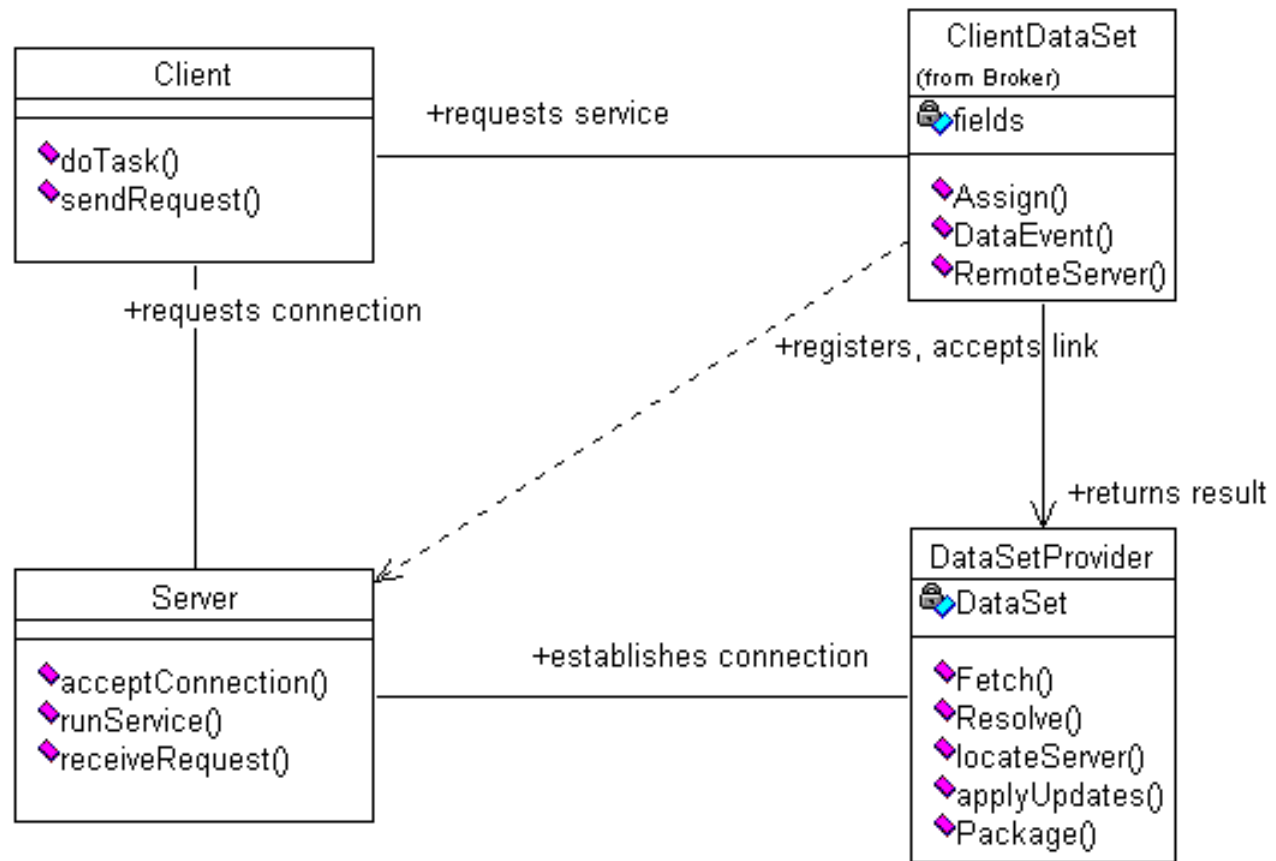
Vorgehen...

MDA Schema



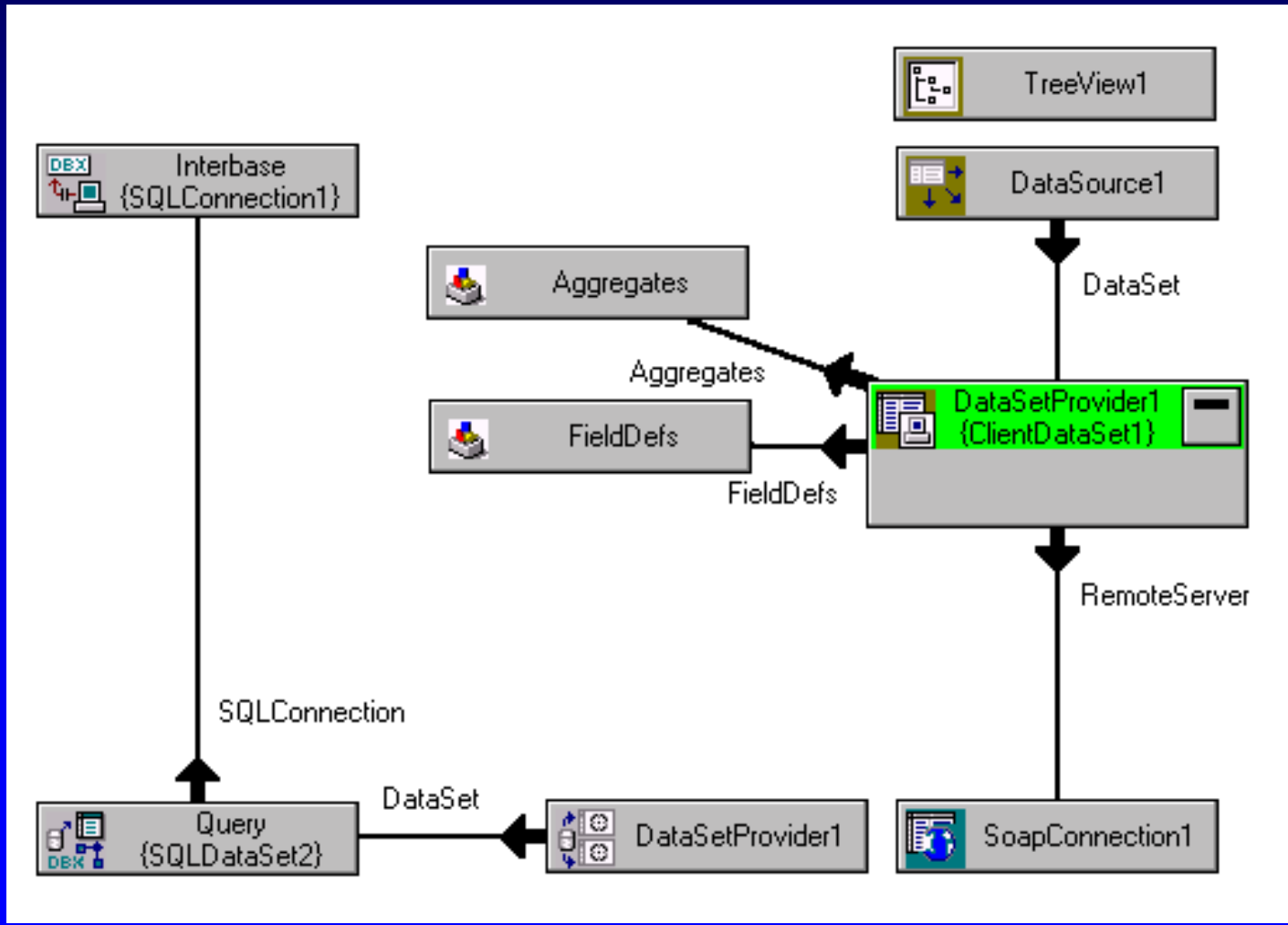


Platform Specific Model





Code auf der Deployment Umgebung





XM Extreme Modelling XM

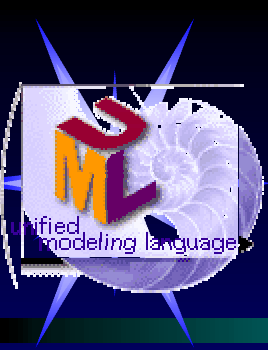
Wozu ein Umstieg ?

- ☯ Dynamische Aspekte wie Zustandsautomaten
- ☯ Kommunikation von Systemkomponenten
- ☯ MDA oder Enterprise Applications Integration
- ☯ Geschäftsprozesse mit viel Aktivitäten <activity>
- ☯ Erstellen von Profilen (OCL, Stereotypen, Values)

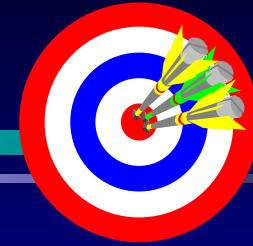
Weshalb nicht?

- ☯ Schon Petrinetze oder Tools wie Simulink existent
- ☯ Entscheid CLX, .NET oder J2EE schon gefällt
- ☯ Toolkette kann fehlen (Debugger, Profiler...)
- ☯ 1.x nicht kompatibel zu 2.0 (nicht verlustfrei)
- ☯ Spezifikationsprache noch nicht standardisiert





All the best!



- ☯ Ab November UML 2.0 Skript auf:
<http://max.kleiner.com/download/umlscript2.zip>
- ☯ Von Viren, Bugs, Würmern und Warteschlangen oder Überleben im digitalen Dschungel

☯ max.kleiner.com

