Entwicklung von Softwaresystemen mit der PAC- und Metalevel-Architektur

Vortrag Robert Bosch Gmbh CM-DI Frankfurt Mai 2002

Burkhardt Renz

Fachhochschule Gießen-Friedberg
Fachbereich Mathematik, Naturwissenschaften und Informatik
Wiesenstr. 14
D-35390 Gießen
Burkhardt.Renz@mni.fh-giessen.de







Entwicklung von Softwaresystemen mit der PAC- und Metalevel-Architektur

Übersicht







Was sind Muster, und was Architekturmuster?

> »Jedes Muster ist eine dreiteilige Regel, die eine Beziehung zwischen einem bestimmten Kontext, einem Problem und einer Lösung beschreibt.« Christopher Alexander

Was macht ein Muster aus?

Ein Muster beschreibt ein in einem speziellen Kontext auftretendes konzeptionelles Problem und präsentiert eine erprobte generische Lösung dafür.

Ein Muster besteht also aus:

- *Kontext*: eine Situation, in der ein Problem auftritt,
- *Problem*: das in diesem Kontext oft entsteht und
- *Lösung*: eine erprobte Auflösung des Problems.







Eigenschaften von Mustern

Muster helfen dabei, die Qualität von Software zu verbessern, weil das Wissen aus guten Architekturen und Entwürfen genutzt wird.

Muster haben die folgenden Eigenschaften:

- Muster dokumentieren bekannte, erprobte Lösungen.
- Muster beschreiben ein gemeinsames Vokabular von Entwurfsprinzipien.
- Muster sind ein Mittel der Dokumentation von Software-Architekturen.
- Muster unterstützen das Erreichen qualitativer Anforderungen.
- Muster reduzieren die Komplexität von Software-Systemen.







Arten von Mustern

Man spricht beim Entwurf von Softwaresystemen von folgenden Arten von Mustern.

Die Begriffe werden jedoch nicht wirklich einvernehmlich verwandt!

- Architekturstil (Shaw/Garlan)
- Architektonischer Mechanismus (Bosch, Kruchten)
- Architekturmuster (Buschmann et al. POSA1)
- Entwurfsmuster (Gamma et al.)
- Sprachidiome (Coplien)







Muster interaktiver und adaptiver Systeme – Von MVC über PAC zur Metalevel-Architektur

Wir wollen ein wohlbekanntes Beispiel betrachten und drei Muster entwickeln, die auf einander aufbauen. Wir beginnen mit dem Standard für interaktive Systeme, erweitern ihn für komplexere Anwendungen und mischen einen Schuss Adaptierbarkeit hinein:

- Model – View – Controller *MVC*
- Presentation – Abstraction – Control *PAC*
- Metalevel-Architektur





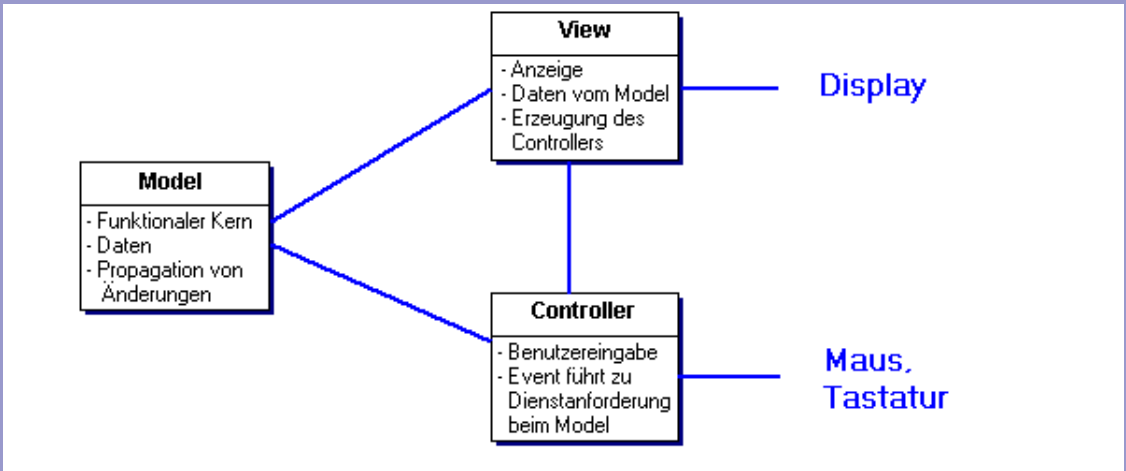


Model – View – Controller *MVC*

Kontext Interaktive Anwendungen mit einer flexiblen Benutzerschnittstelle.

Problem Die Benutzerschnittstelle muss leicht änderbar sein, für bestimmte Benutzer anpassbar sein.

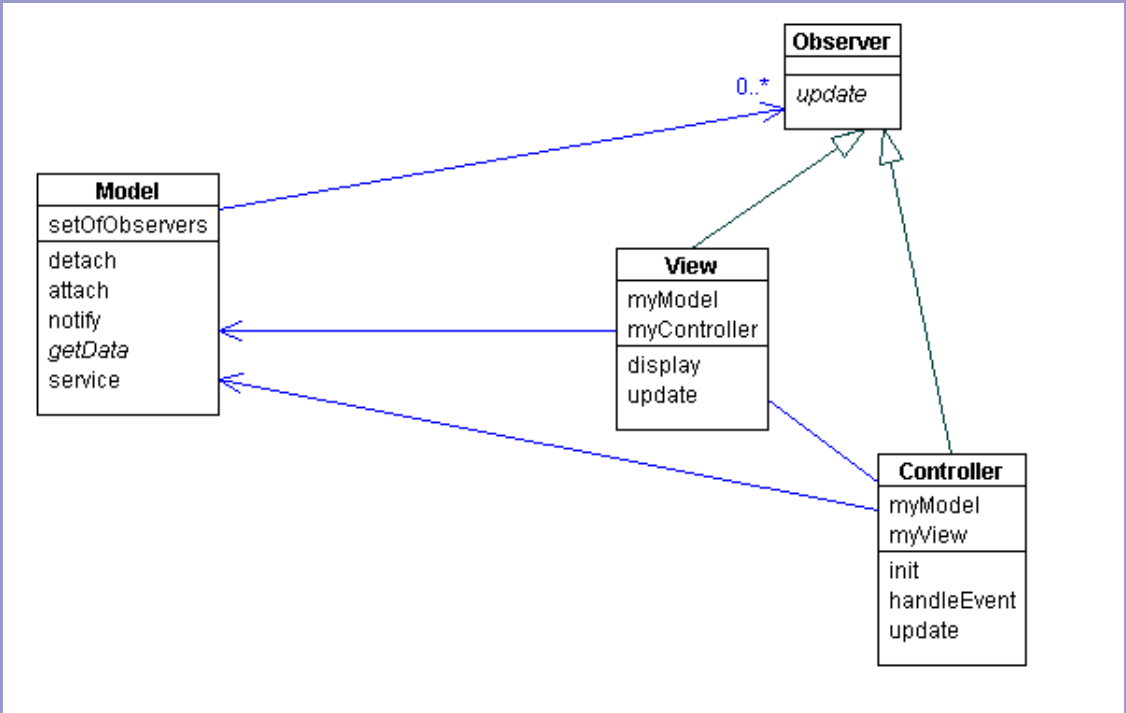
Lösung Unterteilung der Anwendung in drei Komponenten.



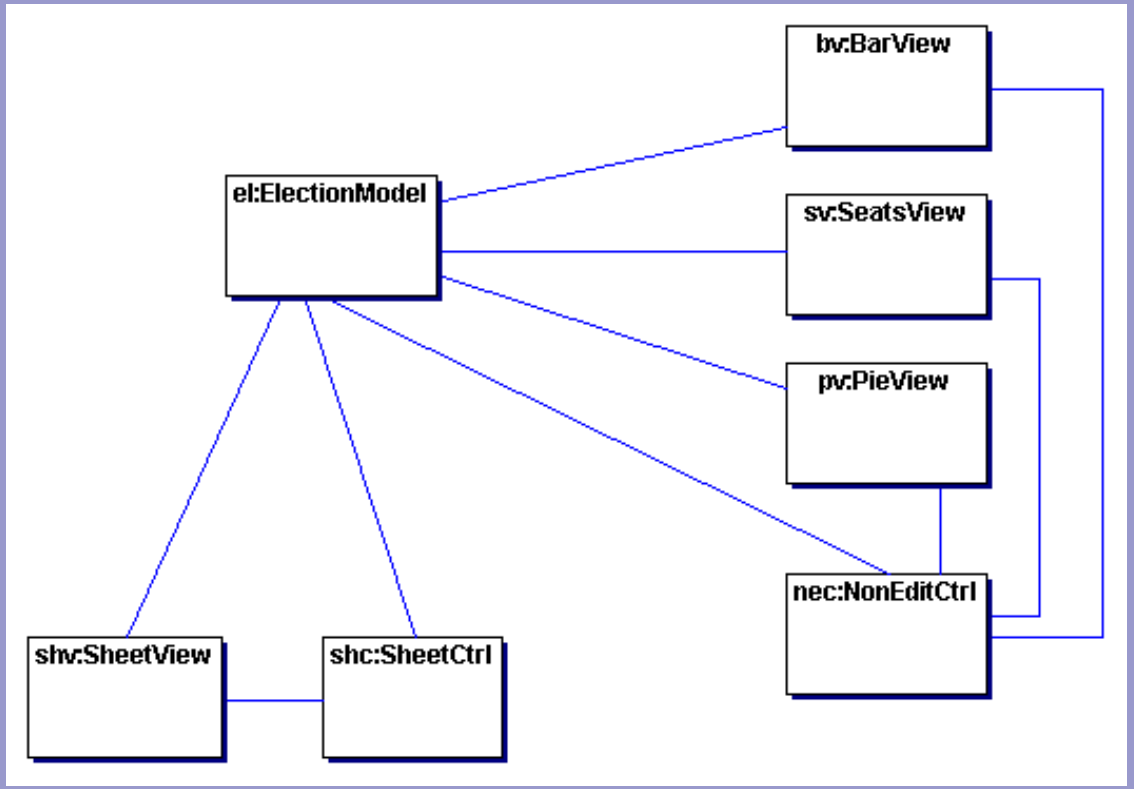
Die Komponenten haben folgende Aufgaben:

- Das *Model* enthält die Funktionalität und die Daten,
- die *Views* präsentieren die Inhalte am Benutzer und
- die *Controler* steuern die Verarbeitung der Eingaben des Benutzers.

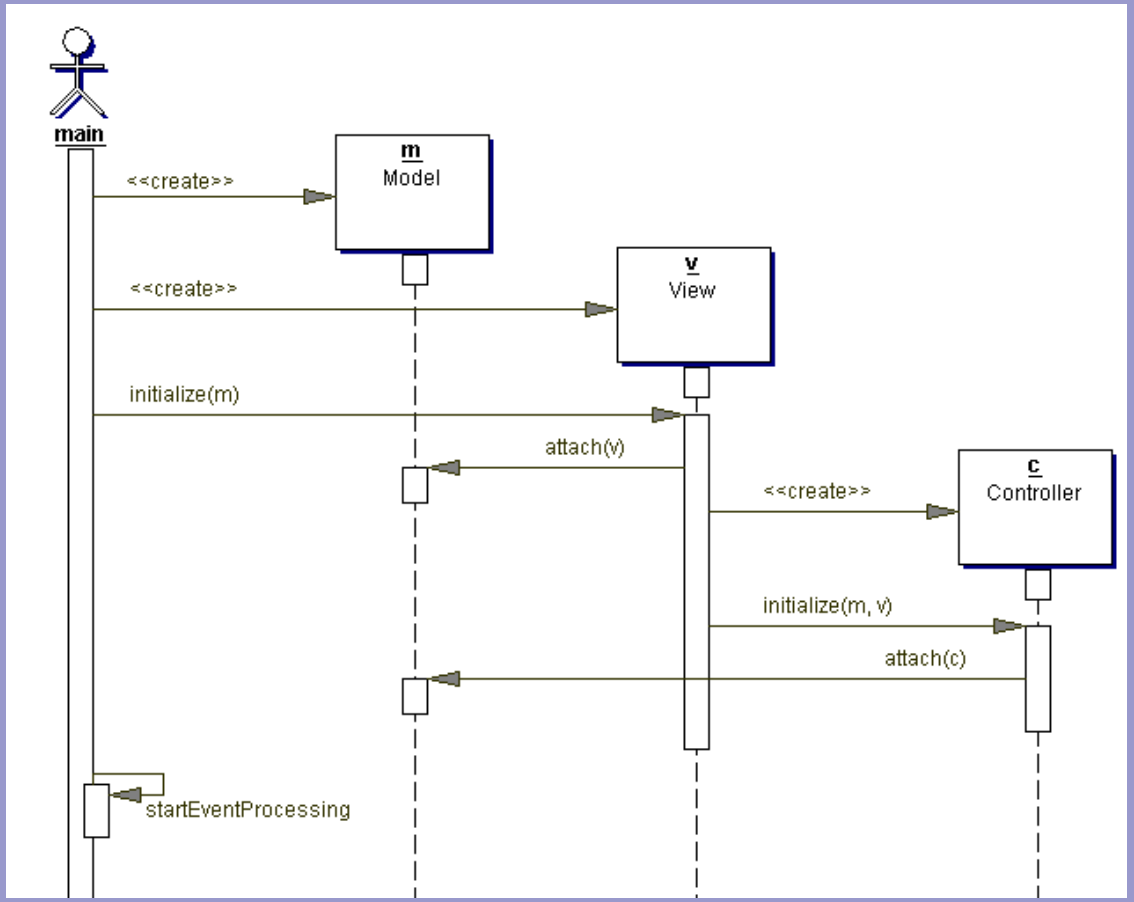
Struktur



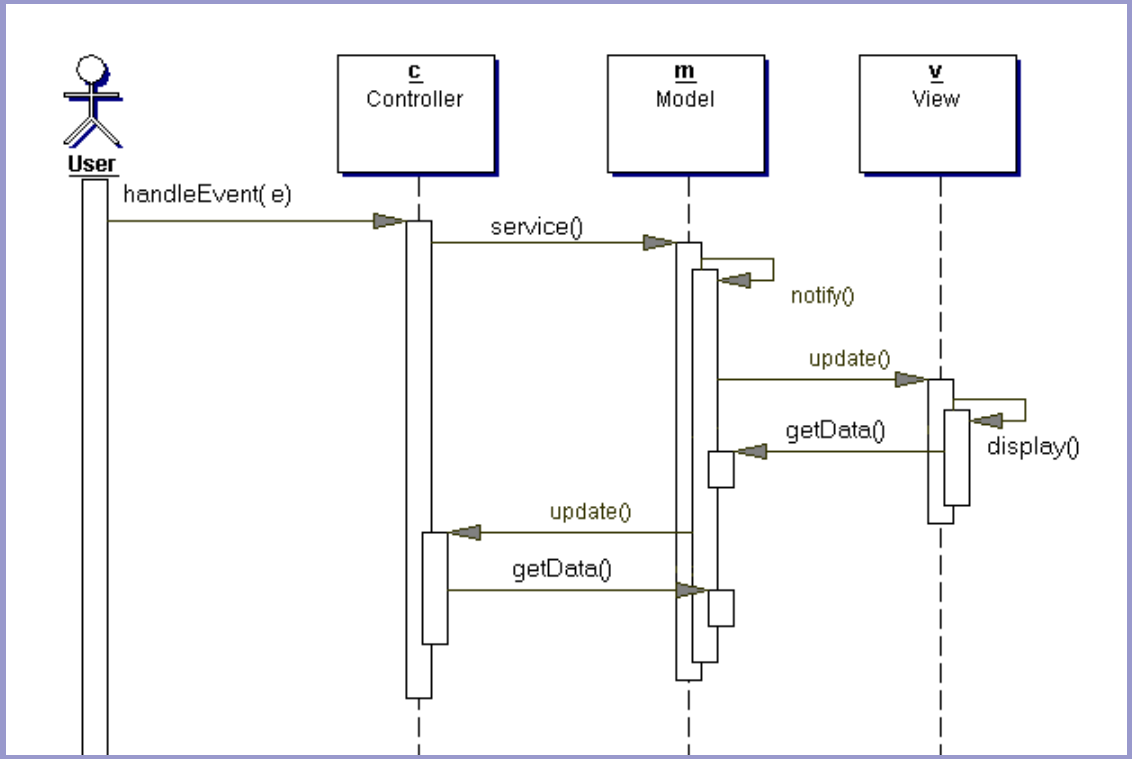
Beispiel Wahlprogramm



Initialisierung von MVC



Dynamik II Reaktion auf eine Benutzereingabe



Varianten und Beispiele Klassisches Beispiel ist *Smalltalk*, von diesem Beispiel stammt das Muster her.

Eine Variante ist die *Document-View-Architektur*, in der View und Controller zusammenfallen und das *Document* die Rolle des Models übernimmt. Prominentes Beispiel sind die *Microsoft Foundation Classes*.

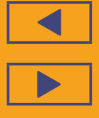
Auswirkungen Vorteile sind

- Mehrere synchrone Ansichten eines Modells.
- »Einsteckbare« Views und Controller.
- Austauschbarkeit von Oberflächenelementen ohne Änderung der Kernfunktionalität.
- Verwendbar in Frameworks.

Nachteile sind

- Komplexität.
- Enge Kopplung von Views und Controllers.
- Eventuell zuviele Aktualisierungen.
- Ineffizienter Datenzugriff in Views.







Presentation – Abstraction – Control *PAC*

Kontext Interaktive Anwendung mit vielfältigen Aufgaben und flexibler Benutzerschnittstelle.

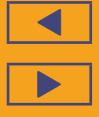
Problem Einheitliche Struktur spezialisierter Agenten und ihrer Zusammenarbeit mit hohem Grad an Änderbarkeit und Erweiterbarkeit.

Agenten verwalten ihren eigenen Zustand und ihre eigenen Daten und kommunizieren mit einem gemeinsamen Protokoll zur Erreichung der Gesamtaufgabe.

Agenten haben spezialisierte Benutzerschnittstelle, entsprechend ihrer Aufgabe.

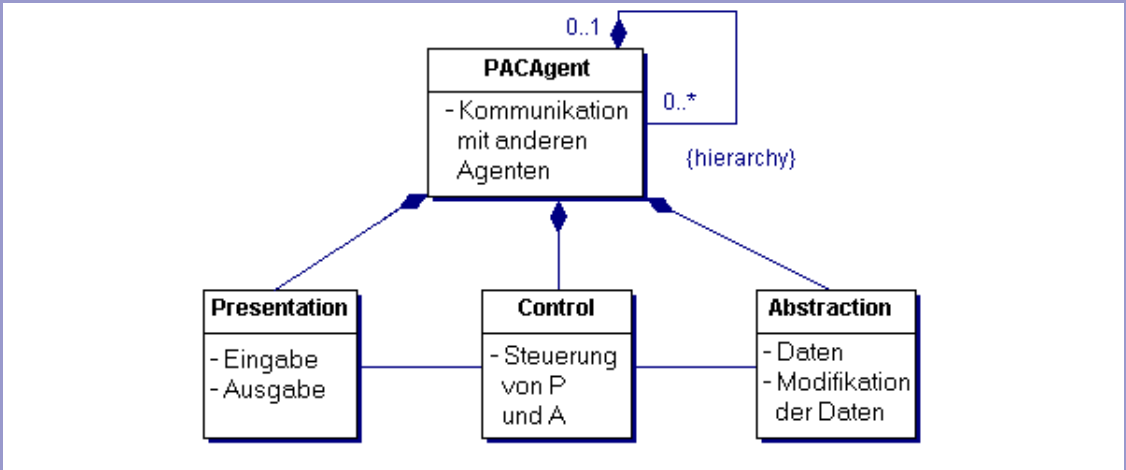
Benutzerschnittstellen ändern sich. Änderungen an einzelnen Agenten oder die Erweiterung des Systems durch neue Agenten soll nicht das ganze System beeinflussen.





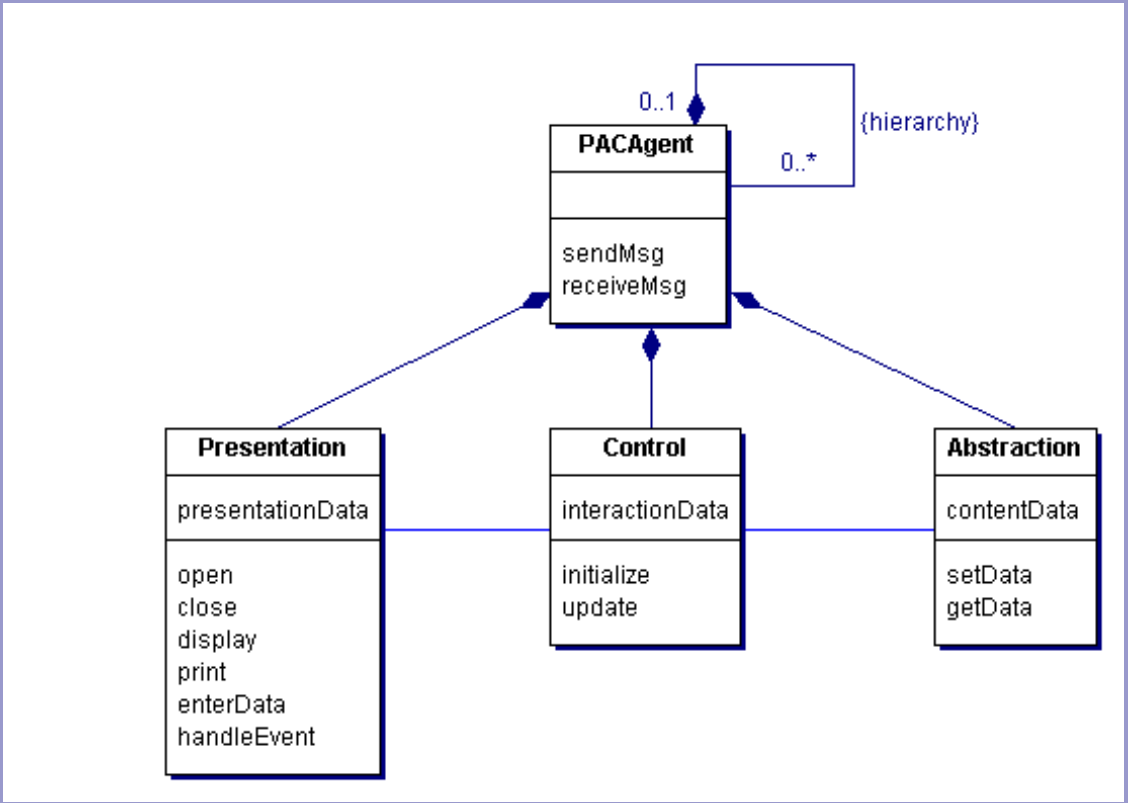


Lösung Hierarchie von Agenten mit drei Komponenten:

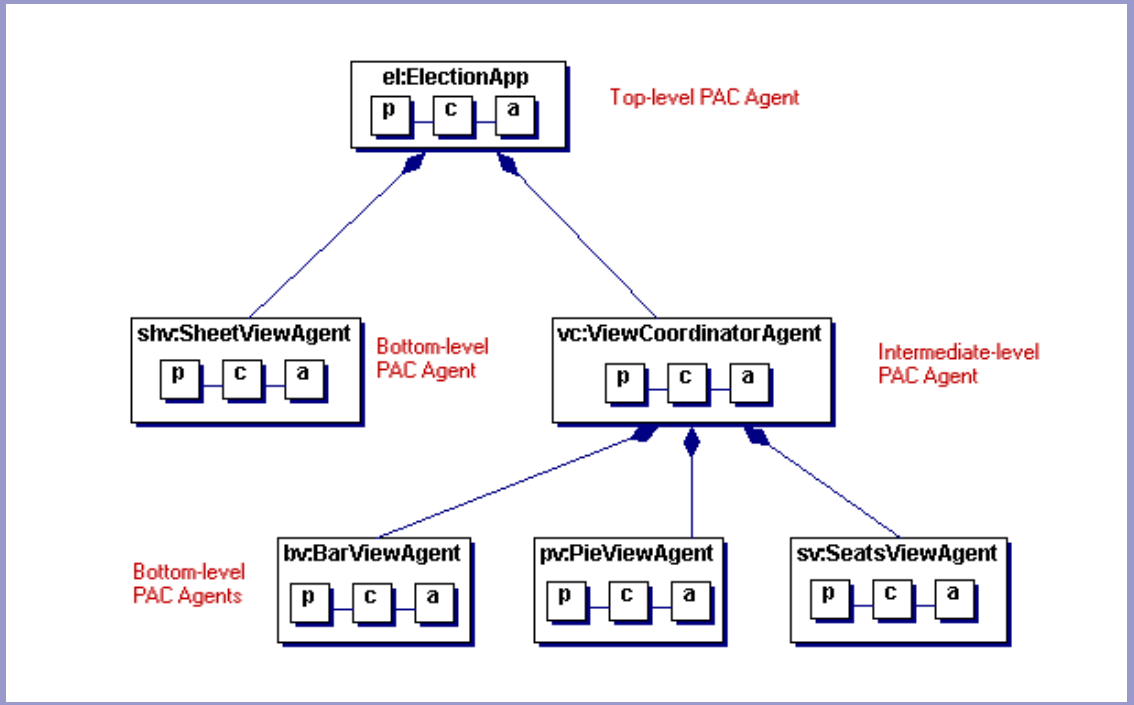


- Hierarchie kooperierender, kommunizierender *Agenten*, die jeweils aufgebaut sind aus Presentation, Abstraction und Control;
- die *Presentation* stellt das sichtbare Verhalten des Agenten dar (Ausgabe und Eingabe),
- die *Abstraction* verwaltet das Datenmodell des Agenten und operiert auf diesen Daten und
- die Komponente *Control* steuert die Verarbeitung zwischen Presentation und Abstraction.

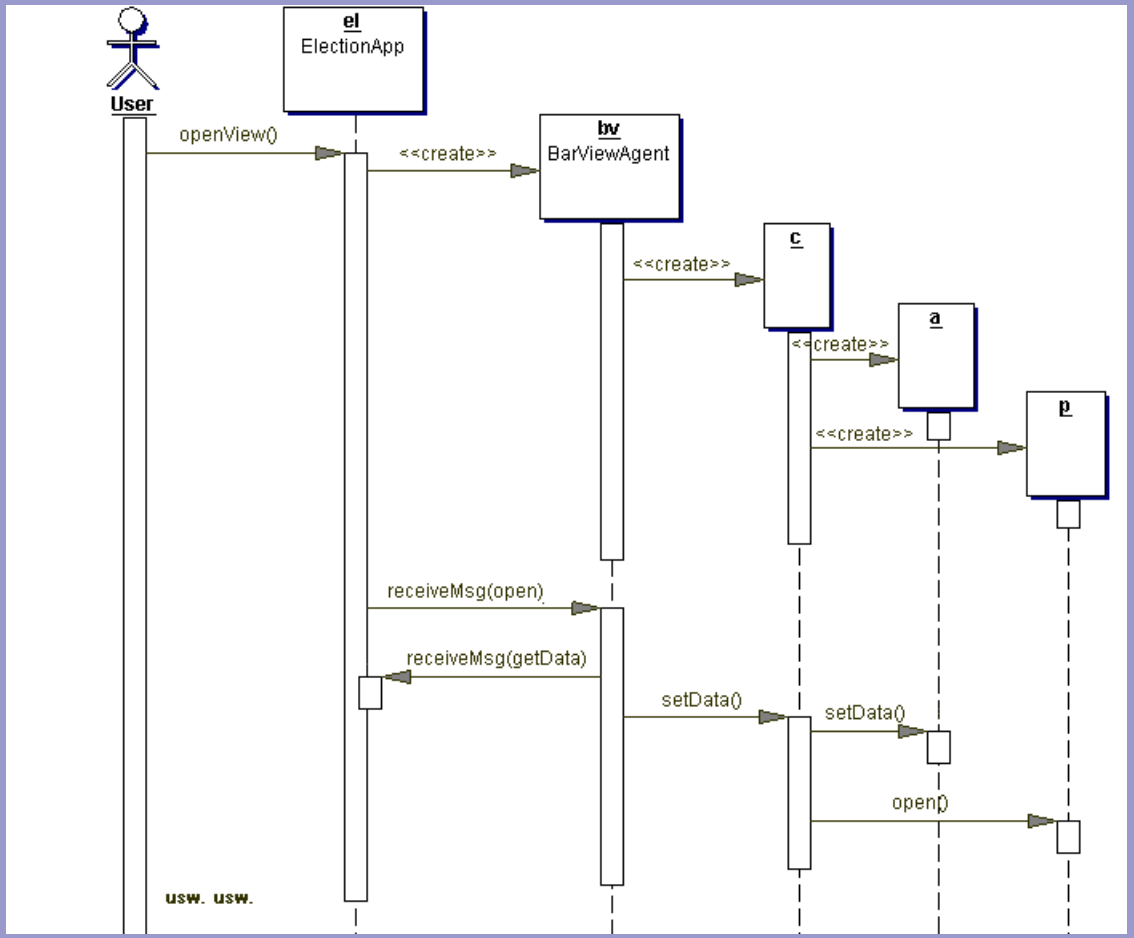
Struktur


0..1
{hierarchy}
0..*
PACAgent
sendMsg
receiveMsg
Presentation
presentationData
open
close
display
print
enterData
handleEvent
Control
interactionData
initialize
update
Abstraction
contentData
setData
getData

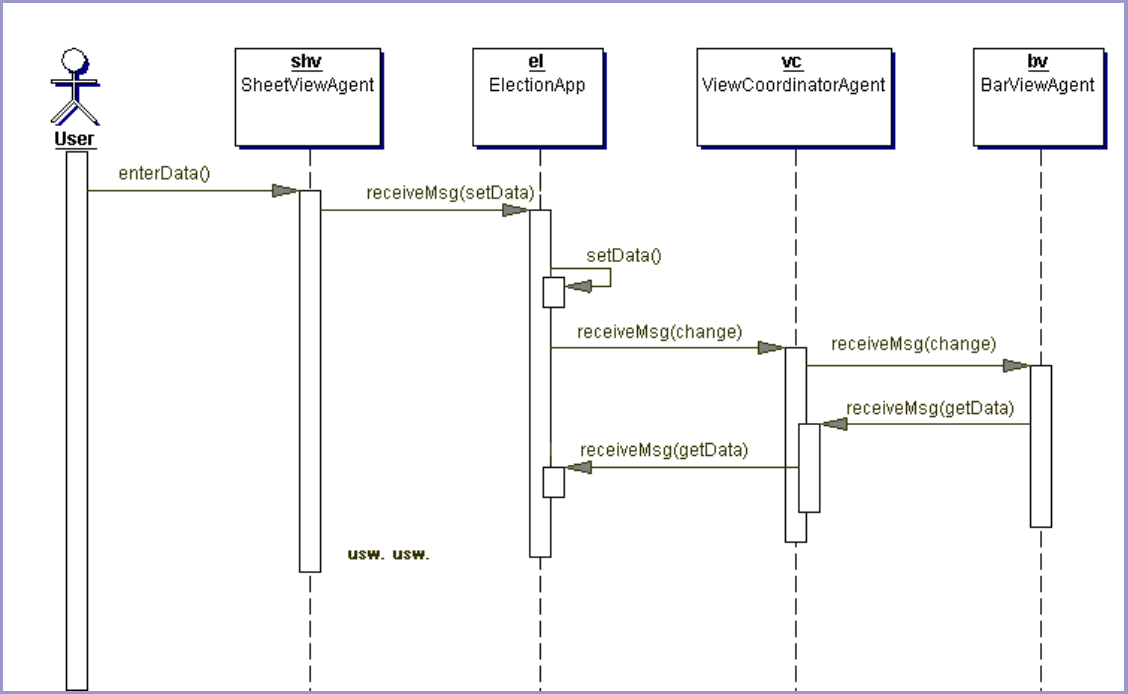
Wieder das Beispiel Wahlprogramm



Dynamik I Initialisierung eines Agenten (BarViewAgent)



Dynamik II Reaktion auf eine Benutzereingabe



Varianten und Beispiele Die Architektur stammt von Joëlle Coutaz (1987) und wird für große spezialisierte Systeme eingesetzt.

Eine Variante ist ein Framework für ein System auf Basis von Windows, in dem die Hierarchie von *Windowselementen* als Hierarchie von Agenten abgebildet wird. Eine solche Architektur erlaubt eine hochgradig interaktive Oberfläche, deren konkretes Verhalten durch Informationen in der *Abstraction* gesteuert wird.

PAC kann mit MVC kombiniert werden, indem ein Agent nicht nur eine Presentation, sondern mehrere hat, die er synchronisiert.

Auswirkungen Vorteile sind

- Trennung der verschiedenen Belange, verschiedene semantische Konzept in spezialisierten Agenten.
- Austauschbarkeit von Agenten, Erweiterbarkeit durch neue Agenten.
- Verwendbar in Frameworks, auch für verteilte Anwendungen.

Nachteile sind

- Hohe Komplexität.







- Komplexe Control-Komponente.
- Agenten müssen Protokoll teilen.
- Hohe Kommunikationskosten in der PAC-Triade und zwischen den Agenten.

Vergleich mit MVC

- Beide trennen funktionellen Kern von Oberfläche.
- PAC steuert die Konsistenz zwischen Daten und Oberfläche durch eigene Komponente.
- PAC ist ein allgemeineres Architekturmuster.







Metalevel Architektur, auch Reflection

Kontext Entwicklung von Systemen, die ihre eigene Adaptierbarkeit von vorneherein unterstützen.

Problem Anwendungen sind auf rasch wechselnde oder optionale Anforderungen nur aufwändig anpassbar, wenn jeweils der Code geändert werden muss.

Lösung

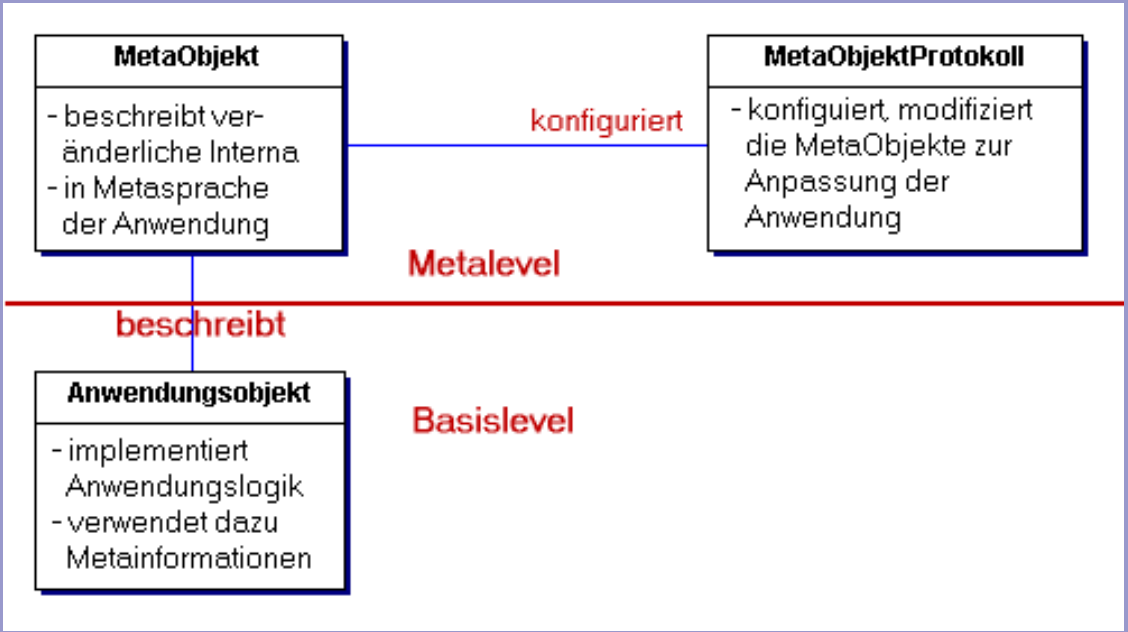
- Trenne die Anwendung in eine Basisebene und eine Metaebene.
- Die Metaebene enthält Metaobjekte, die die Eigenschaften der Objekte der Basisebene beschreiben.
- Die Metaobjekte erlauben die dynamische Veränderungen *der* Eigenschaften der Objekte der Basisebene, die sich oft ändern.
- Wenn ein Objekt der Basisebene verwendet wird, verhält es sich gemäß seiner Beschreibung in der Metaebene.
- Das Metaobjektprotokoll MOP erlaubt den Zugriff auf die Metaebene, es wird durch Experten verwendet.



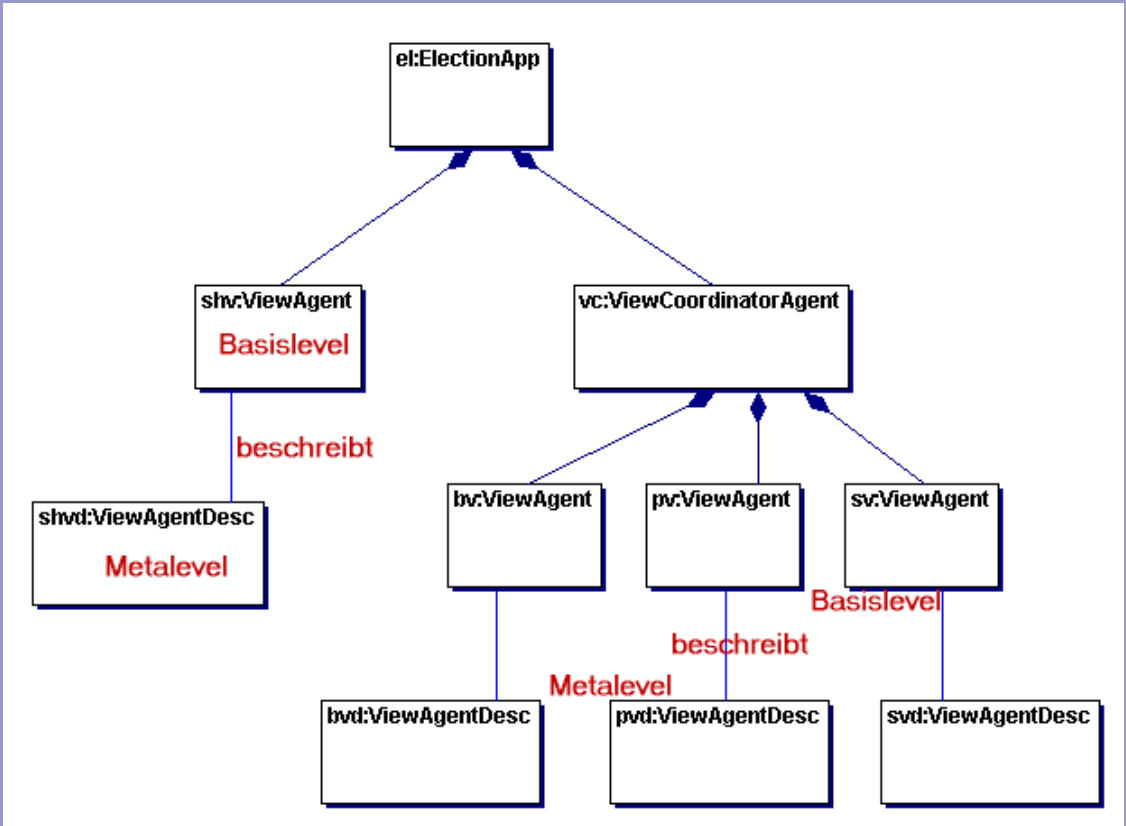




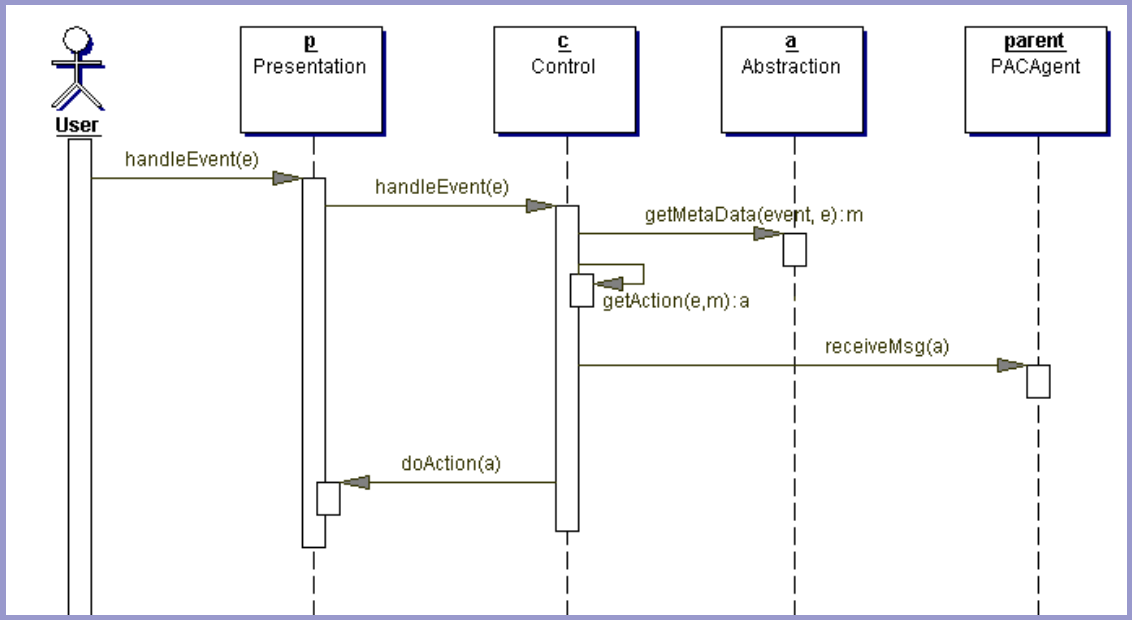
Struktur Die Struktur kann man nur schwer allgemein angeben, deshalb gibt die folgende Graphik mehr eine Idee des Musters:



Objekte Wenden wir die Idee auf unser Wahlprogramm an:



Dynamik Wenn im Beispiel des Wahlprogramms die Metainformation in der Abstraktionskomponente des PACAgenten bei der Erzeugung hinterlegt wird, dann hat man folgende Dynamik innerhalb eines Agenten:



Varianten und Beispiele Variante, die Metaebene nur bei der Konstruktion von Objekten der Basisebene verwendet.

Metaebene wird zum Teil der Abstraction innerhalb eines PACAgenten.

Editoren für die Metaebene, graphische Oberfläche für MOP.

Adaptive Objektmodelle als Technik der Modellierung mit einer Reihe von Analysemustern: *TypeObject*, *RuleObjects*, *Strategy*...

Reflektion in Programmiersprachen wie CLOS, C# und Java.

Viele Spezialanwendungen wie Autohändlersystem NEDIS, Standesamtssoftware Autista, Systeme im medizinischen Bereich.

Auswirkungen Vorteile sind

- Leicht an neue Anforderungen anpassbar.
- Geringere Zahl von Klassen, weil viele Objekte durch Metaobjekte gesteuert werden.
- Änderungen können ohne Kompilieren ins System eingebracht werden.







- Experten des Fachgebiets können über das MOP Änderungen und Anpassungen vornehmen (ohne Änderung des Codes).
- Varianten eines Systems mit dieser Architektur können mit geringem Aufwand erstellt werden.

Nachteile sind

- Architektur schwierig: was wird sich ändern, was nicht?
- Infrastruktur für Metaebene erforderlich.
- Entwicklung aufwändiger.
- Durch zusätzliche Indirektion schwerer zu verstehen und zu warten.
- Hohes Abstraktionsvermögen bei Entwicklern erforderlich.
- Kann mäßige Performance haben.







Beispiel eines interaktiven und adaptierbaren Systems zur Vorgangsbearbeitung

Funktionale und qualitative Anforderungen

Die Anwendung dient der Vorgangsbearbeitung in einer Behörde (Standesamt).

Es werden etwa 50 verschiedene Arten von Vorgängen bearbeitet.

Es gibt sehr detaillierte rechtliche Vorgaben für die Vorgehensweise, diese Vorgaben ändern sich oft, auch kurzfristig; bestimmte Vorschriften sind von Bundesland zu Bundesland verschieden.







Funktionale Anforderungen

- Rechtlich korrekte Bearbeitung von Vorgängen im Amt.
- Hochgradig angepasste Benutzeroberfläche für Anwender verschiedener Kenntnisstufe bis hin zum Experten.
- Expertenmodus in eigener Verantwortung des Beamten.
- Abbildung der Abläufe in der Behörde.

Qualitative Anforderungen

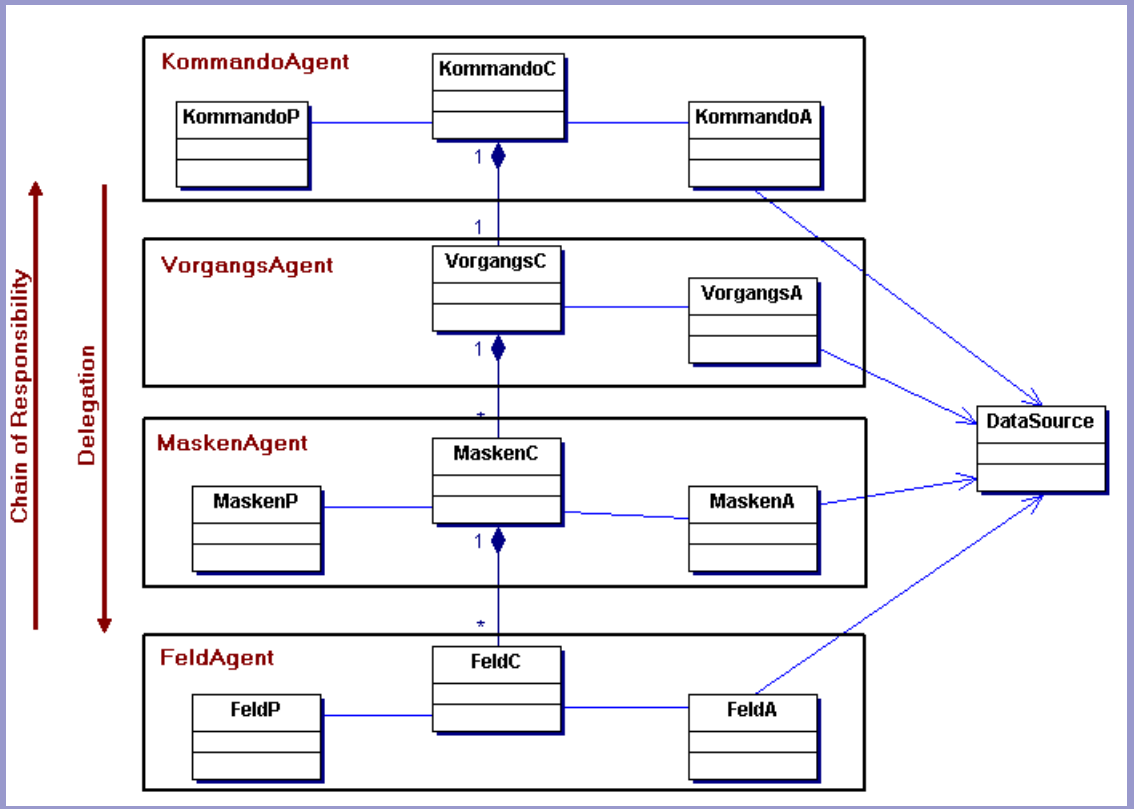
- Rasche Adaptierbarkeit bei rechtlichen Änderungen.
- Anpassbarkeit an regionale und lokale Gegebenheiten.
- Datensicherheit und Datenschutz.
- Performanz.
- Wiederverwendbarkeit von Komponenten für ähnliche Programme (Produktlinie).



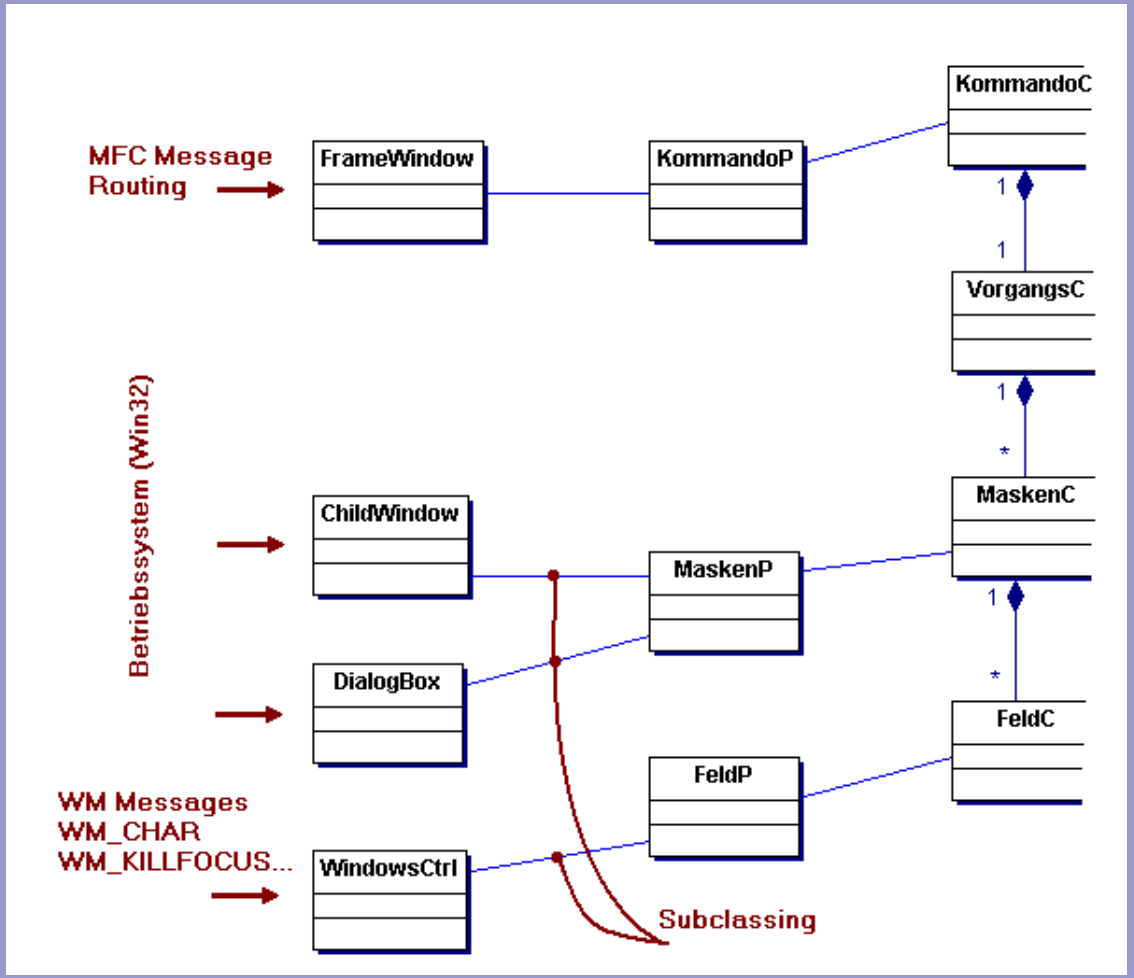




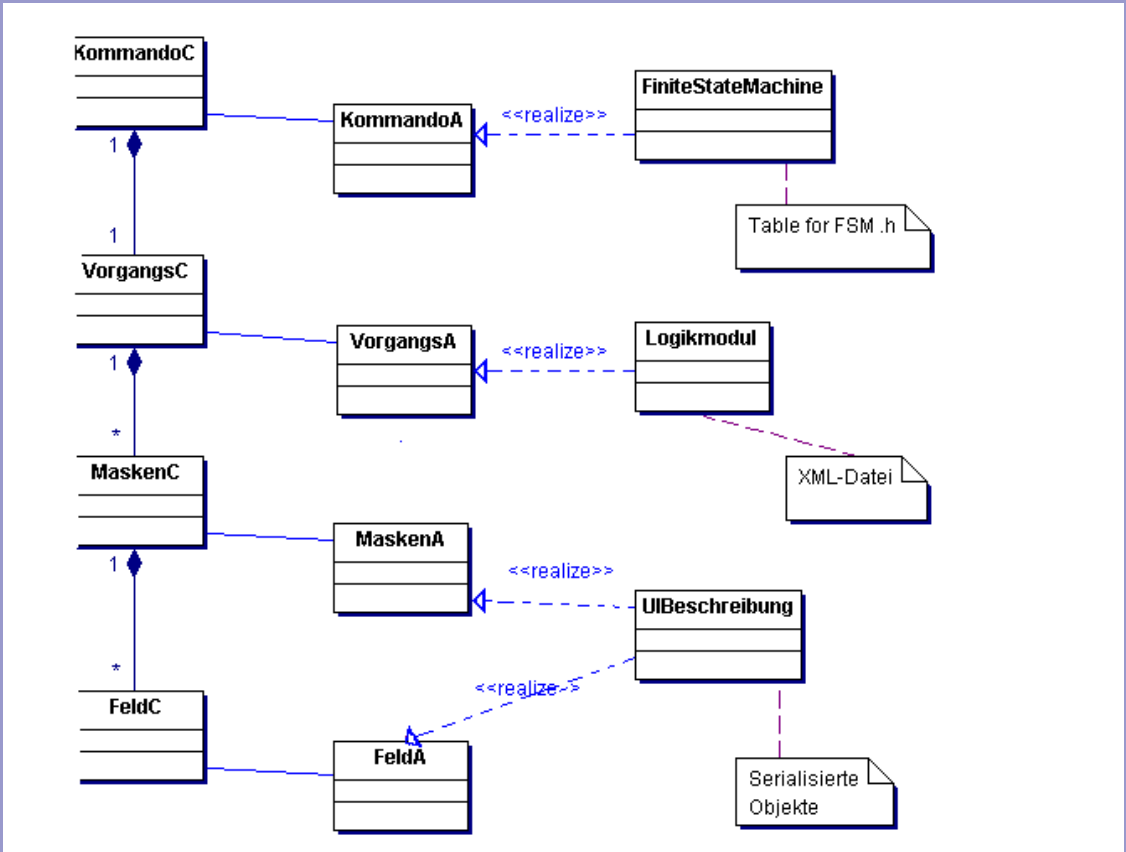
Architektur: PAC plus Metalevel



Kooperation mit dem Betriebssystem Windows



Varianten der Implementierung der Metaebene



Heute würde man anders verfahren:

- Erzeugung der Steuerung für FiniteStateMachine aus UML-Diagramm oder aus XML-Beschreibung (MS Excel nicht geeignet).
- Beschreibung des Logikmoduls nicht durch Ini-Dateien sondern durch XML-Dateien. Durch XSL-Transformationen kann man jede denkbare Form für die Verwendung erzeugen.
- Datenzugriff: Beschreibung der Datenobjekte durch eine XPath-Notation, Zugriff durch XPath oder XQuery. Werkzeuge heute sehr weit entwickelt, siehe auch Microsoft ADO.NET
- Erweiterung der Metaprotokoll-Ebene denkbar durch das Interception-Muster: man sieht vor, dass der Zugriff auf die Metaebene durch Interceptors aufgefangen und so adaptiert werden kann (siehe [4]).







Konsequenzen für Qualitätsmerkmale

Performanz: je höher der Grad der Adaptierbarkeit (Zeitpunkt!?) sein soll, desto höher ist die Einbuße an Performanz. Andererseits: Varianten können sehr schnell sein.

Speicherbedarf: »Redundanter« Code entfällt, Beschreibungen können zur Laufzeit geladen werden.

Wartbarkeit: Änderungen entlang der Adaptionsstellen sind sehr leicht möglich. Andererseits: Designzentren kaum änderbar.

Testbarkeit: PAC und Metalevel hat klare Schnittstellen und Protokolle, die gut testbar sind. Andererseits: Sprachen für den Metalevel können für Überraschungen sorgen.

Erweiterbarkeit: gut durch Einbauen neuer Agenten, gut durch Erweiterung der Fähigkeiten des Metalevels. Andererseits: schlecht gegen die Adaptionslinien.

Aufwand: sehr hoch; lohnt nur, wenn Adaptierbarkeit ein Muss ist oder Wiederverwendung in Produktlinie vorgesehen ist.







Literatur

[1] **Mary Shaw, David Garlan** *Software Architecture: Perspectives on an Emerging Discipline*, Upper Saddle River, New Jersey: Prentice-Hall, 1996.

[2] **Jan Bosch** *Design and Use of Software Architectures*, Harlow, England: Addison-Wesley, 2000.

[3] **Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal** *Pattern-orientierte Software-Architektur: Ein Pattern-System*, Bonn: Addison-Wesley-Longman, 1998.

[4] **Douglas Schmidt, Michael Stal, Hans Rohnert, Frank Buschmann** *Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects*, Chichester: John Wiley, 2000.

[5] **Michael Jackson** *Problem frames: Analysing and structuring software development problems*, Harlow, England: Addison-Wesley, 2000.

[6] **Joseph W. Yoder, Federico Balaguer, Ralph Johnson** *Architecture and Design of Adaptive Object-Models*, Web: `www.joeyoder.com`, 2001.