

# Agenten erhöhen Wartbarkeit

## Eine agentenbasierte Softwarearchitektur für grafische Benutzungsoberflächen



Dr. Jörg-Volker Müller

LINEAS Informationstechnik GmbH

<http://ccds.lineas.de>



## CompetenceCenter Distributed Solutions

- ▶ LINEAS
  - ▶ Gründung 1989
  - ▶ Holding-Struktur
  - ▶ > 200 Mitarbeiter
  - ▶ 7 Unternehmen mit unterschiedlichen Aufgabenbereichen
- ▶ CompetenceCenter Distributed Solutions
  - ▶ Geschäftsbereich der LINEAS Informationstechnik GmbH
  - ▶ Gründung 1999
- ▶ Fokus: Verteilte und Mobile Unternehmensanwendungen
- ▶ Claim: Wir mobilisieren Ihre Geschäftsprozesse!



- ▶ Kontext
  - ▶ integral.dX
  - ▶ integral.dX GUI
- ▶ integral.mca
  - ▶ Anforderungen
  - ▶ Architektur
- ▶ APIs für GUIs
  - ▶ Model-View-Controller (MVC)
  - ▶ Presentation-Abstraction-Control (PAC)
- ▶ integral.mca API
  - ▶ Services
  - ▶ Properties
- ▶ Anwendungsbeispiel: Calculator
- ▶ Konsequenzen



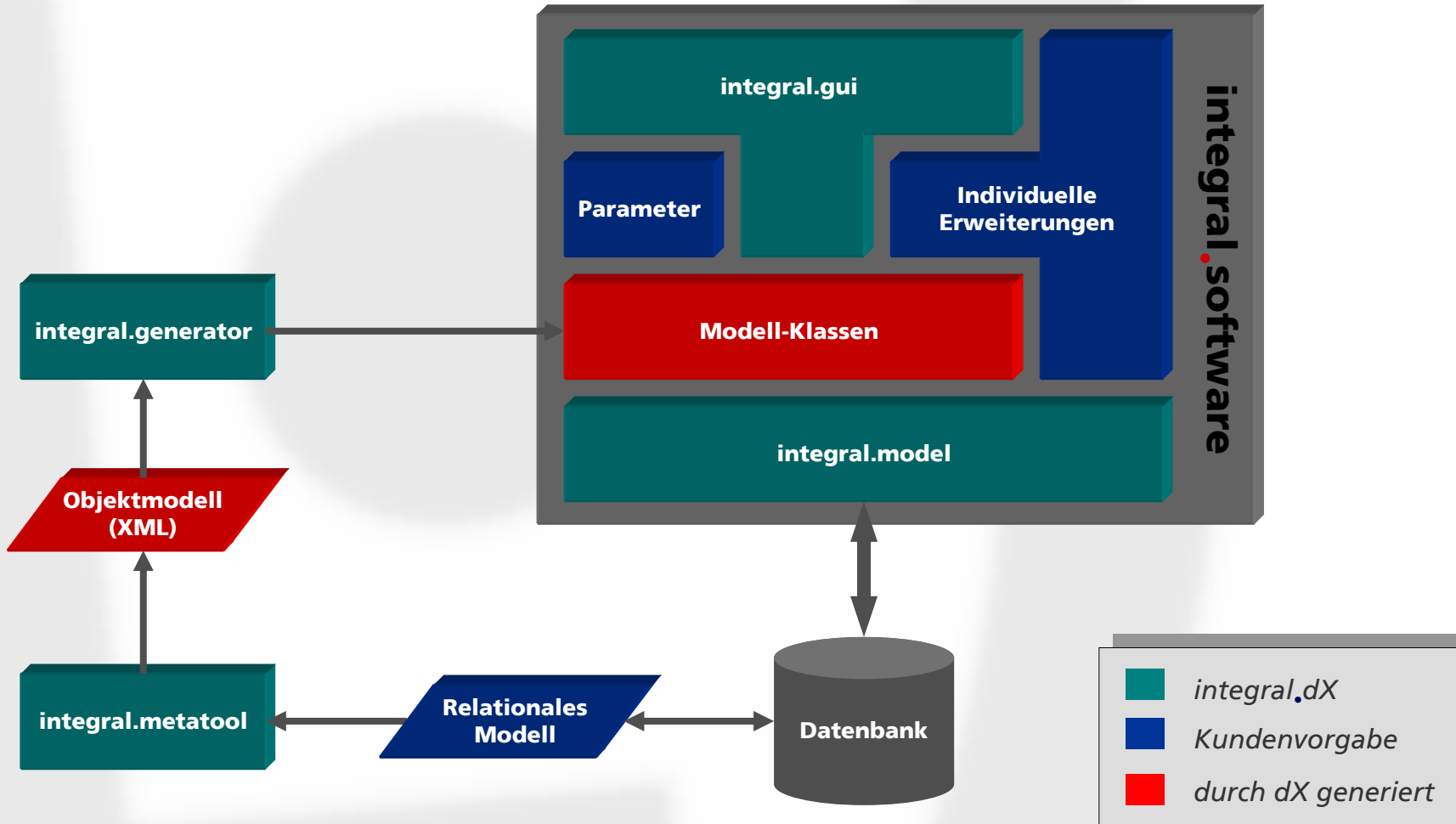
# Kontext

## integral.dX – Model

- ▶ Softwarearchitektur integral.dX für die Realisierung datenbankbasierter Softwarelösungen
- ▶ deskriptive Mengen erlauben eine Navigation innerhalb des Datenmodells
- ▶ Datenbankmapping-Schicht mit
  - ▶ **getyptem Zugriff**  
`kunde.getName()`
  - ▶ **Metadaten**  
`Kunde.type.NameDecl.getType()`
  - ▶ **generischem Zugriff**  
`kunde.get(Kunde.type.NameDecl)`
- ▶ ermöglicht beispielsweise einen generischen Algorithmus zum kaskadierenden Löschen



- ▶ Automatischer Aufbau der grafischen Oberfläche aus
  - ▶ Metadaten
  - ▶ Konfigurationsparametern (vgl. ini-Datei)
- ▶ Vorteile
  - ▶ Einfache, durchgängige Datenpflegeoberfläche
  - ▶ Masken haben einheitliches Look&Feel und Bedienkonzept
  - ▶ Aus-/Einblenden von Feldern per Konfiguration möglich
  - ▶ Benutzerspezifisches Layout per Konfiguration realisierbar
  - ▶ Layout kann gerätespezifisch (Bildschirmgröße, Farben, Stifteingabe etc.) variieren



integral.time.admin 2.3.5

Feiertage (22)  
Funktionen (11)  
Kategorien (22)  
Kostenstelle (0)  
Kunden (15)  
Mandant (0)  
Mitarbeiter (22)  
Mitarbeitertypen (5)  
ProjectTypeFunction (11)  
Projekte (24)  
Projekte:Angebote (0)  
Projekte:Bestellungen (0)  
Projekte:Produktivität (3)  
Projekte:Rechnungen (0)  
Projekte:Status (8)  
Projekte:Typen (4)  
Rollen (1)  
Termine (111)  
Urlaubskonto (388)  
Zugriffsprofil (1)  
Überstundenkonto (1771)  
Überstundenmodelle (1)

Mitarbeiter

Nachname: Baumann  
Vorname: Paul  
Mail:  
Personalnummer:  
Einstellungsdatum: 01.01.2002  
Austrittsdatum:  
Mitarbeitertyp: Externe(r)  
Kostenstelle:

Projekte als Mitarbeiter | Termine | Urlaubskonto | Überstundenkonto

Login | Vertragsdaten | Buchungen | Projekte als Leiter

Buchungen	Bemerkung	Dauer	Start	Ende	Datum	Buchungsart
		1			25.01.2002	Krank
		390	13:00	19:30	07.01.2002	Arbeitszeit
		240	8:00	12:00	07.01.2002	Arbeitszeit
		1			29.01.2002	Krank
		1			30.01.2002	Krank
		240	8:00	12:00	08.01.2002	Arbeitszeit
		300	14:00	19:00	08.01.2002	Arbeitszeit
		1			28.01.2002	Krank
		1			24.01.2002	Krank

Datensätze : 22



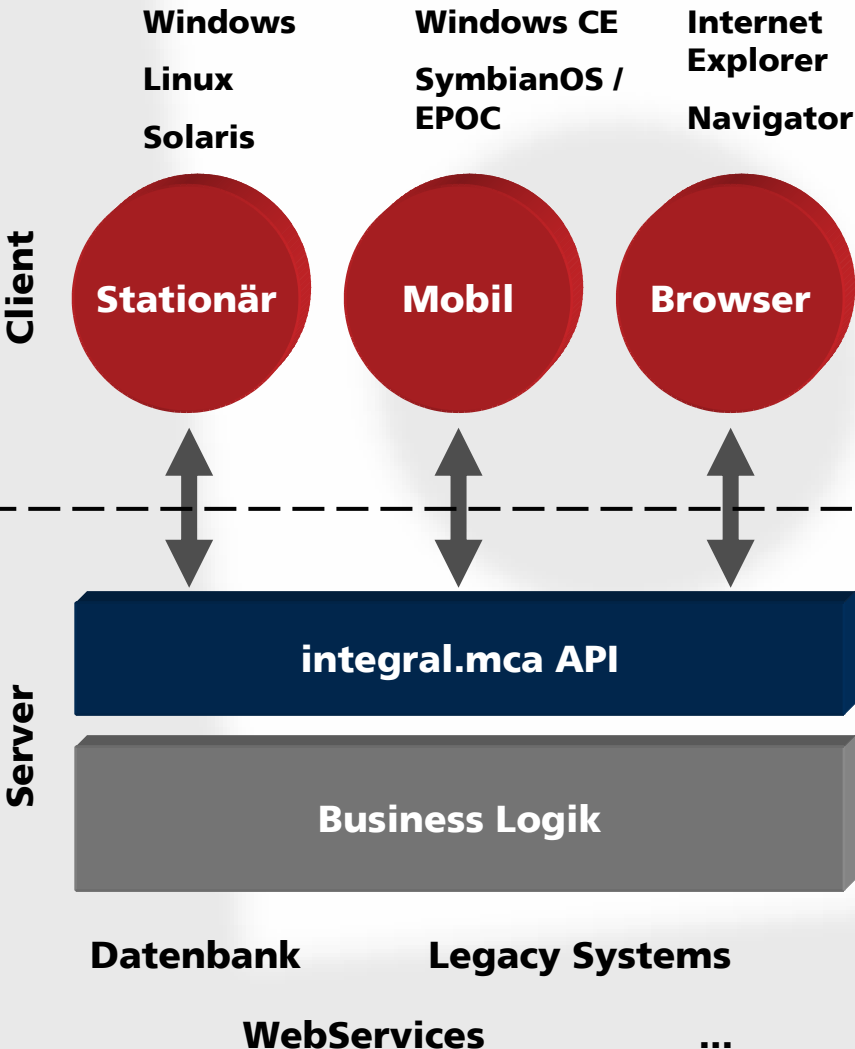
- + mächtig
- + viele Oberflächenelemente
- + leicht zu bedienen
- speicherintensiv
- sehr groß und komplex
- niedriges Abstraktionsniveau
- unnötiger Ballast (Pluggable Look&Feel)
- Look&Feel nicht so gut wie etwa Windows
- langsam

- ▶ Kontext
  - ▶ integral.dX
  - ▶ integral.dX GUI
- ▶ integral.mca
  - ▶ Anforderungen
  - ▶ Architektur
- ▶ APIs für GUIs
  - ▶ Model-View-Controller (MVC)
  - ▶ Presentation-Abstraction-Control (PAC)
- ▶ integral.mca API
  - ▶ Services
  - ▶ Properties
- ▶ Anwendungsbeispiel: Calculator
- ▶ Konsequenzen

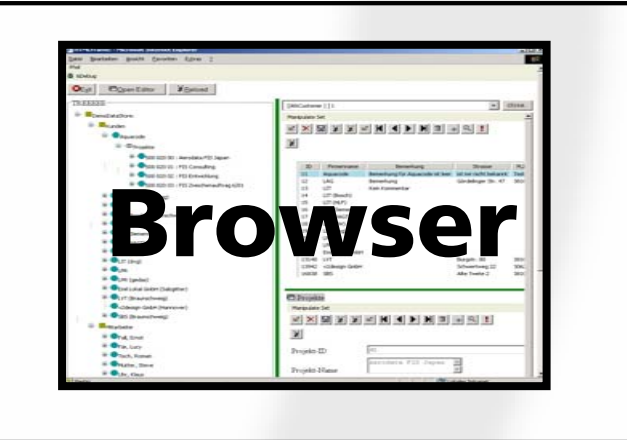
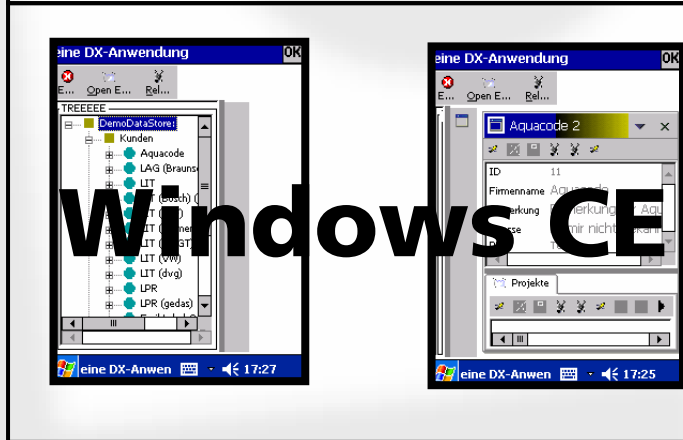
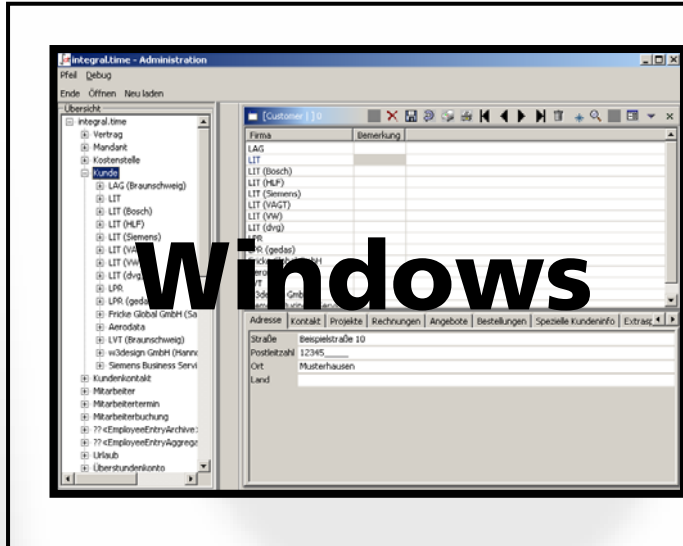
- ▶ 5/2000-8/2000 Erste Untersuchungen
  - ▶ PAC-Agenten
  - ▶ Java Beans und IDE-Einbindung
- ▶ 12/2000 Vortrag GI Regionalgruppe
- ▶ 9/2000–3/2001 Diplomarbeit MCI
  - ▶ Vorstufe: Verteilungsaspekt
  - ▶ Thin-Client-Konzept umgesetzt
- ▶ Frühjahr 2001 Vorphase
  - ▶ Antrag bei Bezirksregierung
  - ▶ Förderung im Rahmen der FTE-Richtlinie der EU
  - ▶ Förderhöhe 50 %
- ▶ 7/2001–11/2002 Realisierung
  - ▶ Erweiterung/Verfeinerung der Thin-Client-Anbindung
  - ▶ Aufbau der serverseitigen Programmierschnittstelle

# integral.mca

## Architektur



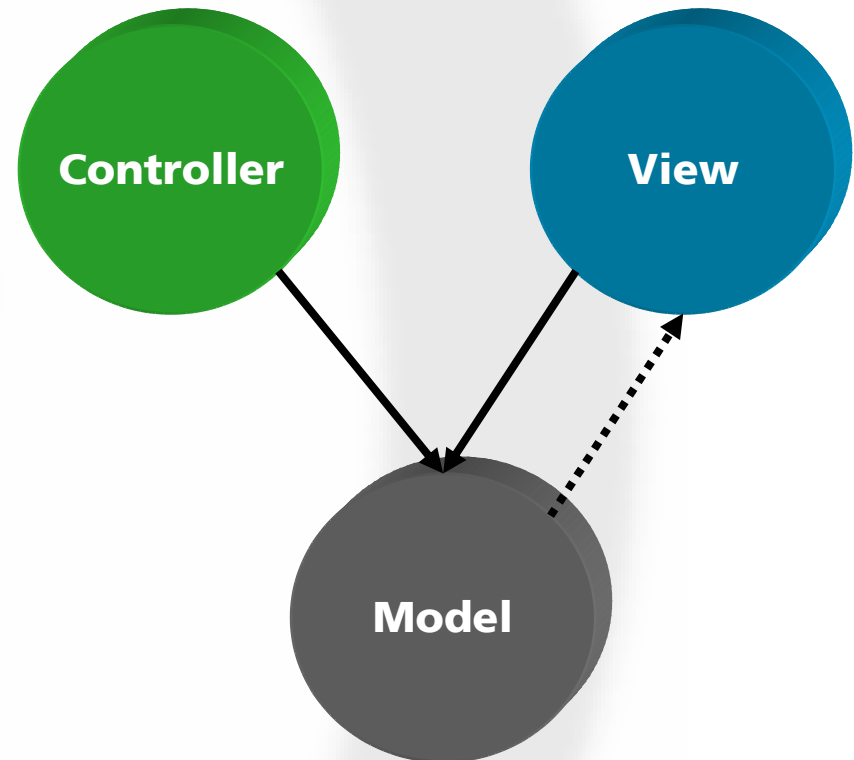
- ▶ **FrontEnd**
  - ▶ plattformoptimiert
  - ▶ anwendungsunabhängig
  - ▶ Lokale Hardware integrierbar (Barcodeleser)
- ▶ **Protokoll**
  - ▶ optimiert für
    - ▶ niedrige Bandbreite
    - ▶ hohe Latenzzeit
  - ▶ Verschlüsselung per SSL
- ▶ **Business Logik**
  - ▶ Realisierung in Java
- ▶ Reduziert den Aufwand in Entwicklung und Betrieb



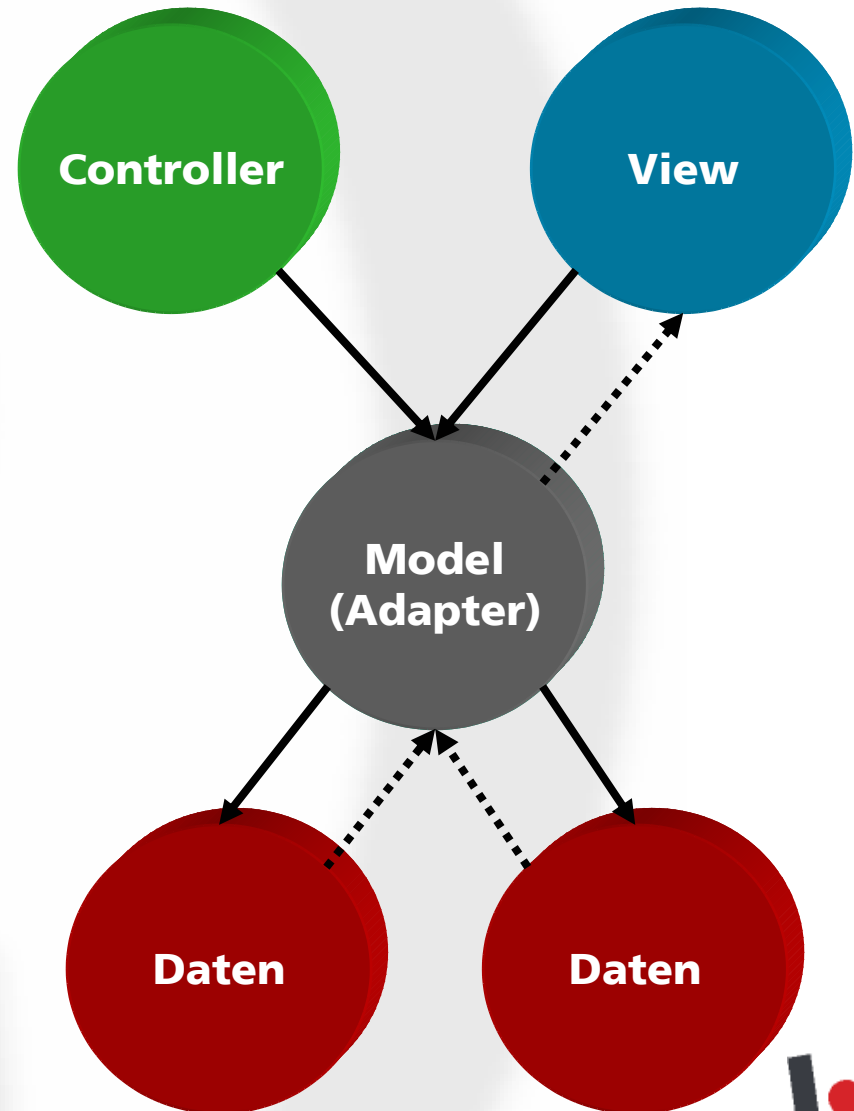
- ▶ Kontext
  - ▶ integral.dX
  - ▶ integral.dX GUI
- ▶ integral.mca
  - ▶ Anforderungen
  - ▶ Architektur
- ▶ APIs für GUIs
  - ▶ Model-View-Controller (MVC)
  - ▶ Presentation-Abstraction-Control (PAC)
- ▶ integral.mca API
  - ▶ Services
  - ▶ Properties
- ▶ Anwendungsbeispiel: Calculator
- ▶ Konsequenzen

## Model-View-Controller – Konzept

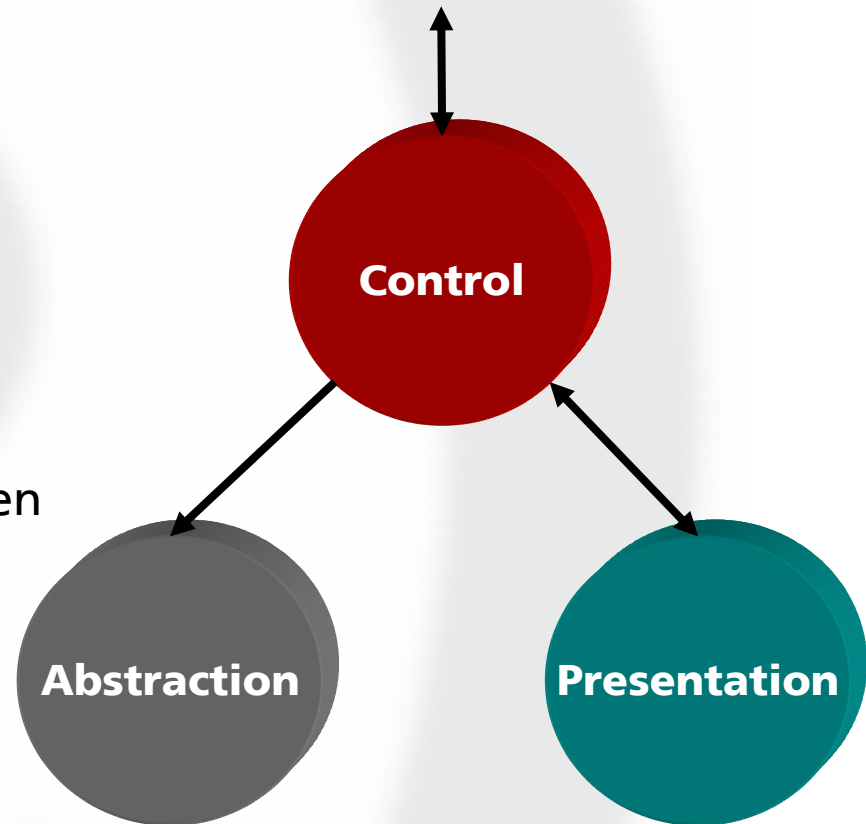
- ▶ **Model**
  - ▶ enthält/kennt die Daten
  - ▶ kennt C/V nicht
  - ▶ benachrichtigt View bei Änderungen
  - ▶ kann mehrere C/V haben
- ▶ **View**
  - ▶ zeigt Daten an
  - ▶ kennt Model
- ▶ **Controller**
  - ▶ steuert die Benutzereingaben
  - ▶ kennt Model
- ▶ Passive Kopplung des Models an Anwendung per get/set



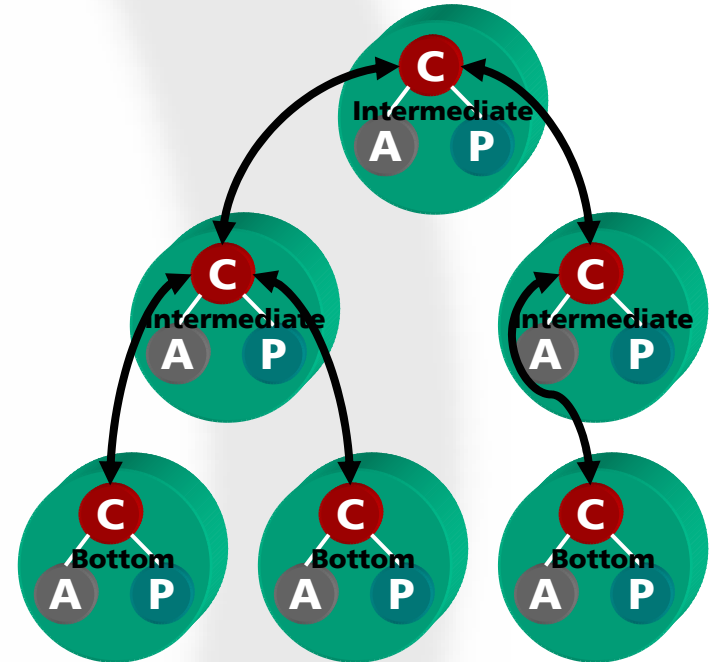
- ▶ Model kann den Datenwert enthalten
- ▶ Typisch in Smalltalk:
  - ▶ Model ist Adapter, der die Daten bereitstellt
- ▶ Konsequenz:
  - ▶ Hohe Flexibilität (Baukastenkonzept)
  - ▶ leicht aufzubauen, schwer abzubauen
  - ▶ unübersichtliche Struktur
  - ▶ komplexes Objektgeflecht
  - ▶ Fehlersuche aufwändig



- ▶ **Presentation**
  - ▶ Darstellung
  - ▶ Benutzereingaben
- ▶ **Abstraction**
  - ▶ hält Daten
- ▶ **Control**
  - ▶ Verbindung A-P
  - ▶ Kommunikation nach außen



- ▶ Man spricht von „Agenten“
- ▶ Agenten bilden eine Hierarchie
  - ▶ Top-Agent
  - ▶ Intermediate-Agent (= Container)
  - ▶ Bottom Agent
- ▶ Agenten kommunizieren über standardisierte Protokolle miteinander (synchronized)
- ▶ Vorteile:
  - ▶ klare Struktur (streng hierarchisch)
  - ▶ einfache Kommunikationsregeln
- ▶ Nur für größere Komponenten geeignet



# APIs für GUIs

## Was sind eigentlich Agenten?

- ▶ Nicholas R. Jennings: "An agent is an encapsulated computer system that is situated in some environment and is capable of flexible, autonomous action in that environment in order to meet its design objectives."
- ▶ Eigenschaften von Agenten
  - ▶ klare Abgrenzung (Kapselung/Schnittstellen)
  - ▶ Beziehung zu ihrer Umgebung
  - ▶ zu erfüllende Ziele
  - ▶ Autonomie
  - ▶ Flexibilität, Fähigkeit zur Adaption



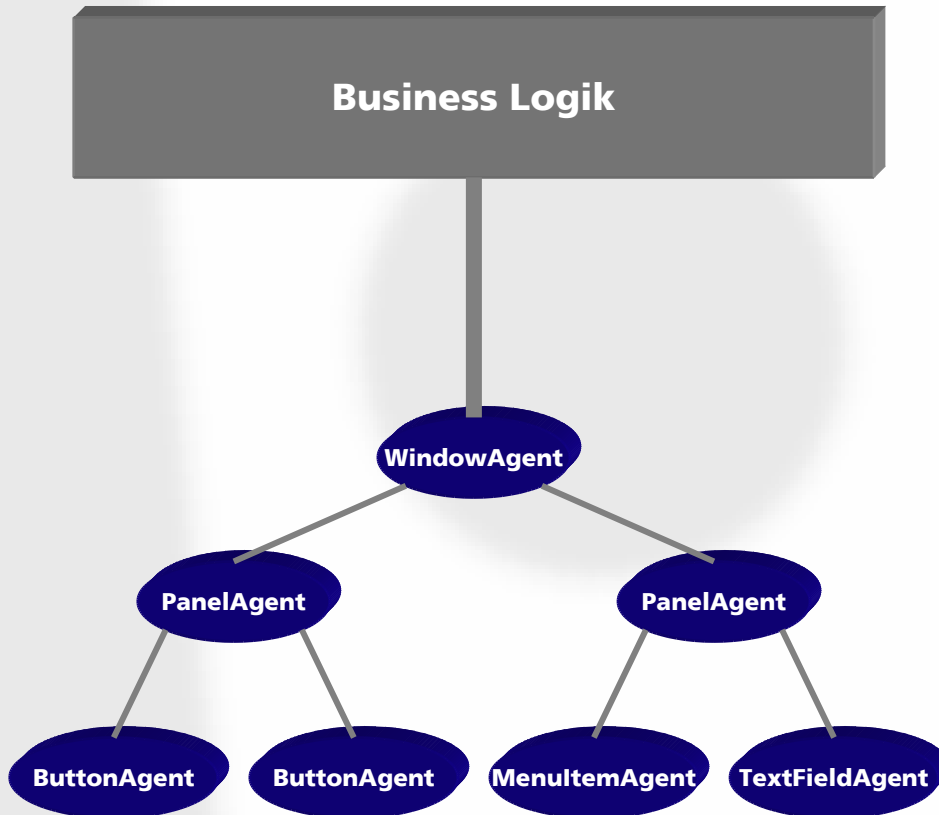
# APIs für GUIs

## Kann man so etwas für GUIs verwenden?

- ▶ klare Abgrenzung
  - ▶ notwendig, gerade das war bei MVC das Problem
- ▶ Beziehung zur Ihrer Umgebung
  - ▶ sinnvoll, wenn grundsätzlich Regeln vorhanden sind, wie der Agent mit der Umwelt kommuniziert
- ▶ zu erfüllende Ziele
  - ▶ hier geht es doch nur um die Darstellung einer Oberfläche?
- ▶ Autonomie
  - ▶ "magisches" Verhalten in der Oberfläche ist dem Benutzer suspekt
  - ▶ deterministisches Verhalten ist gefordert
- ▶ Flexibilität, Fähigkeit zur Adaption
  - ▶ Flexibiliät ist gut, aber selbstständige Adaption?



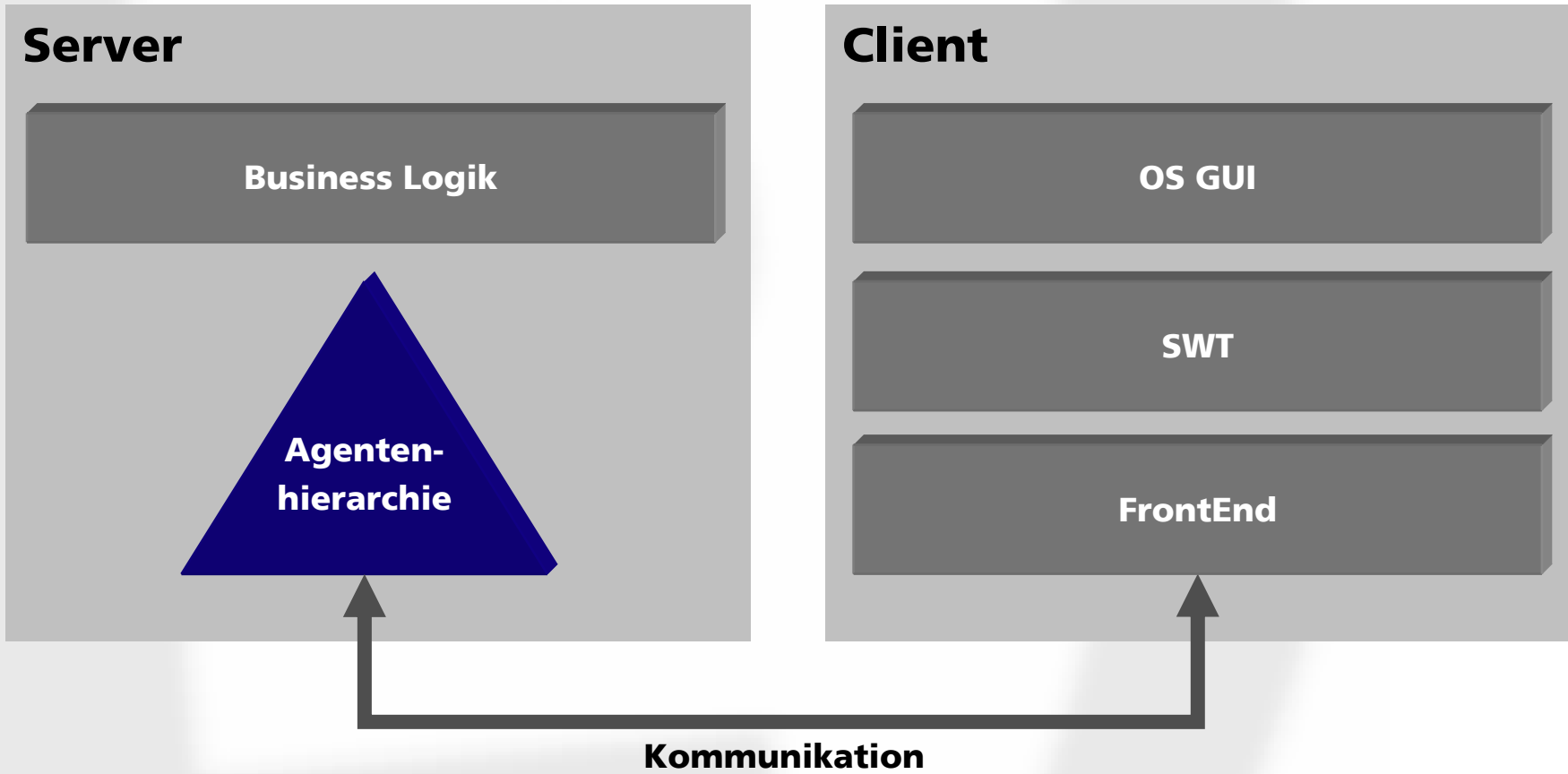
- ▶ Kontext
  - ▶ integral.dX
  - ▶ integral.dX GUI
- ▶ integral.mca
  - ▶ Anforderungen
  - ▶ Architektur
- ▶ APIs für GUIs
  - ▶ Model-View-Controller (MVC)
  - ▶ Presentation-Abstraction-Control (PAC)
- ▶ integral.mca API
  - ▶ Services
  - ▶ Properties
- ▶ Anwendungsbeispiel: Calculator
- ▶ Konsequenzen

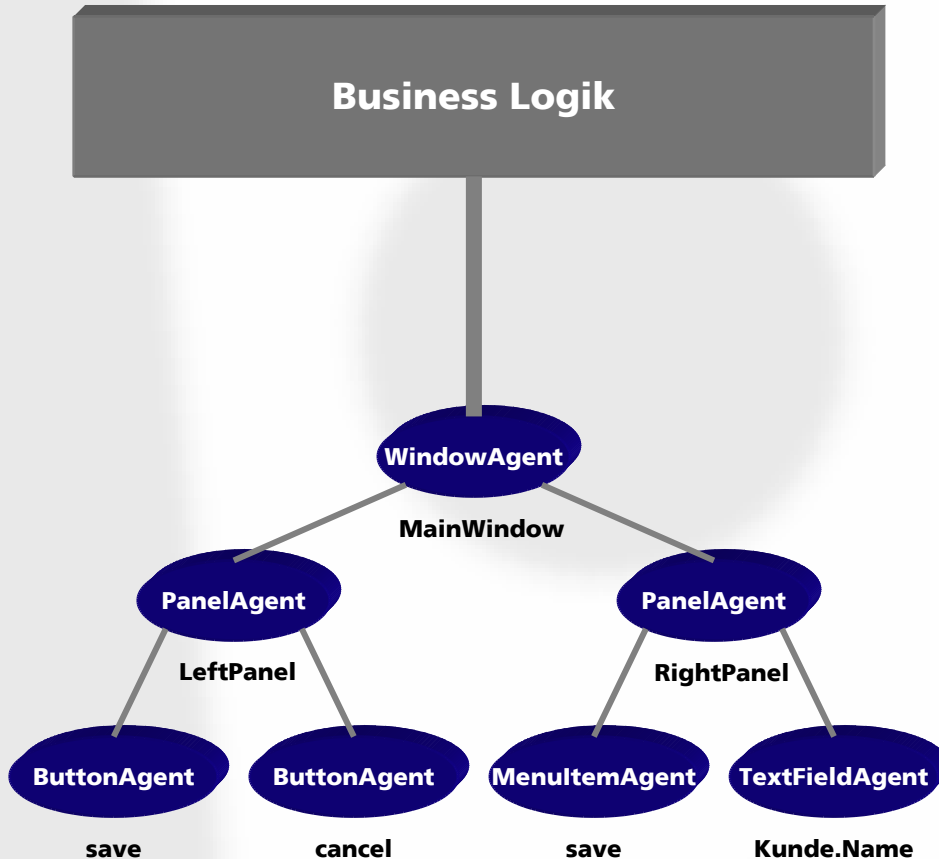


- ▶ Agenten bilden Hierarchie von GUI Komponenten
- ▶ Hierarchie entspricht der Aufteilung der GUI
- ▶ Aufbau durch Programmierung (IDE nicht realisiert)
- ▶ Typische Agententypen:
  - ▶ Buttons, Labels
  - ▶ Eingabefelder
  - ▶ Tabelle, Baum
  - ▶ Panels, Windows
- ▶ Business Logik ist praktisch RootAgent

# integral.mca API

## The Big Picture

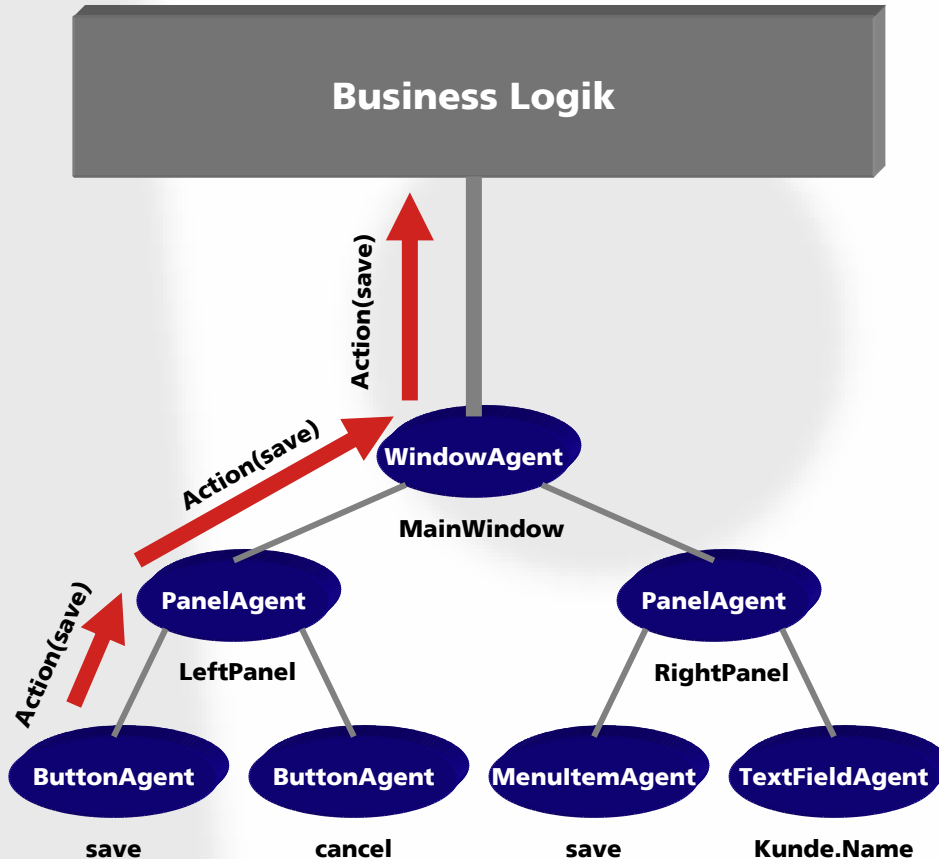




- ▶ Jedem Agenten wird bei der Konstruktion ein *Aspekt* mitgegeben
- ▶ Aspekte sind praktisch beliebige Objekte, die der Agent als Schlüssel nutzt
- ▶ Agenten und Business Logik kennen sich gegenseitig nicht

# integral.mca API

## Services – Upstream



- ▶ Agenten stellen Services bereit (Root/Intermediate)
- ▶ Agenten fordern Services an (Intermediate/Bottom)
- ▶ Beispiel: Button "save" wurde gedrückt

- ▶ Services werden deklariert
  - ▶ bereitgestellte Services werden registriert
  - ▶ Benutzung wird angezeigt (required/optional)
- ▶ statische Überprüfung nicht möglich
- ▶ Typsicherheit ist gewährleistet
- ▶ Dynamische Überprüfung von Bereitstellung und Benutzung bei Konstruktion des Baums
  - ▶ Werden alle Required Services auch bereitgestellt?
- ▶ bei Modifikation des Baums zur Laufzeit erneute Überprüfung
- ▶ verschiedene Optimierungen sorgen für performanten Betrieb

# integral.mca API

## Services – Upstream

### ⚡ Bereitsteller (Anwendung)

```
registerService (  
    MCAStandardServices.onAction,  
    saveAspect,  
    saveProcessor );
```

### ⚡ Nutzer (Button)

```
ServiceHandle actionHandle = addRequired (  
    MCAStandardService.onAction,  
    this.selfAspect );
```

### ⚡ alternativ

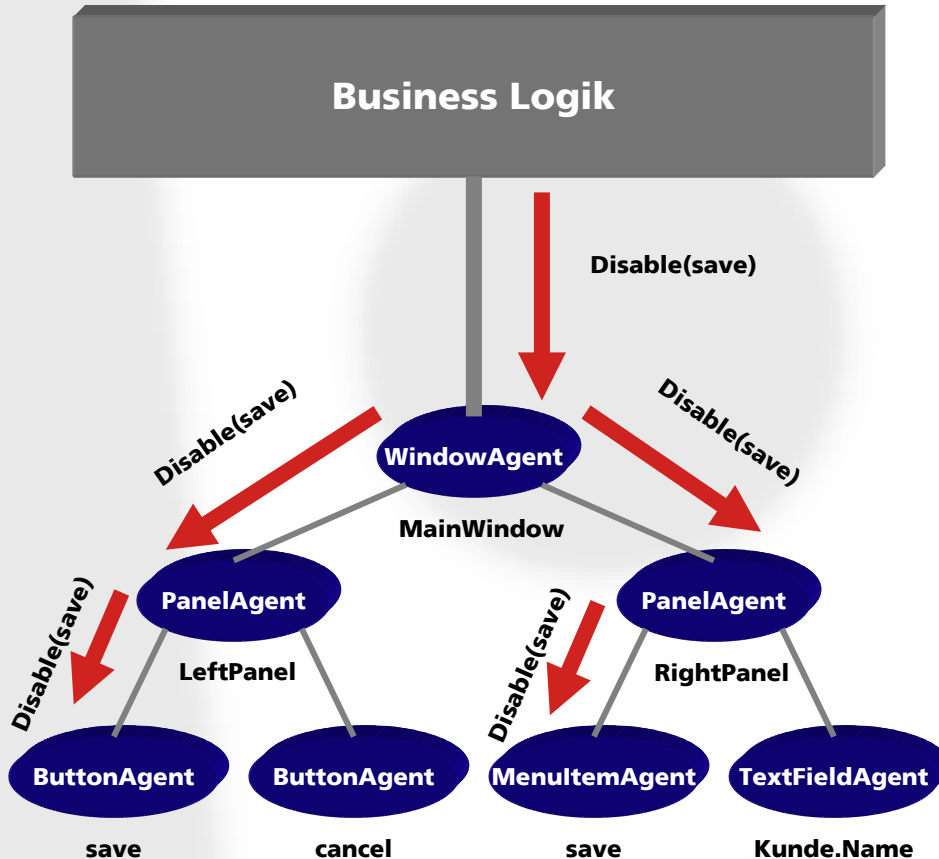
```
ServiceHandle traceHandle = addOptional (  
    MCA StandardService.tracelogAction,  
    this.selfAspect );
```

### Aufruf

```
Boolean result = invoke (actionHandle);
```

# integral.mca API

## Services – Downstream



- ▶ Auch die andere Richtung wird unterstützt
- ▶ Dynamische Switching-Tabellen wissen, wo welche Agenten sind
- ▶ Beispiel: Disable "save"

# integral.mca API

## Services – Downstream

### ⚡ Nutzer (Anwendung)

```
ServiceHandle disableHandle = addRequired (  
    MCAStandardService.disableAction,  
    saveAspect );
```

### ⚡ alternativ

```
ServiceHandle disableOptHandle = addOptional (  
    MCA StandardService.disableAction,  
    saveAspect );
```

### Benutzer

```
Boolean result = invoke(disableHandle);
```

### ⚡ Bereitsteller

```
registerService (  
    MCAStandardServices.onAction,  
    this.selfAspect,  
    disableProcessor );
```



- ▶ Zugriff über Services?
  - ▶ ↑ getValue(someAspect)
  - ▶ ↑ setValue(someAspect)
  - ▶ ↓ valueChanged(someAspect)
  - ▶ Zwischenspeicher im benutzenden Agenten
- ▶ Hoher Implementierungsaufwand
- ▶ Hohe Fehlerwahrscheinlichkeit (z. B. Tippfehler)
  
- ▶ Konsequenz: Properties

- ▶ Properties haben eine get- und eine set-Methode
- ▶ Properties bieten einen Listener-Mechanismus für Wertänderungen
- ▶ Properties cachen den Wert

### ▶ ⓘ Bereitstellen einer Property

```
registerProperty ( booleanGlobalProperty, nameAspect,  
    new AbstractPropertyProcessor() {  
        protected Object get(Aspect aspect) {  
            return propertyValue;  
        }  
        protected void set(Aspect aspect, Object newValue) {  
            handlePropertyChange(newValue);  
        }  
    }  
);
```

### ↑? Nutzer (Button, MenuItem)

```
PropertyHandle valueHandle = addRequired (  
    MCAStandardService.StringPropertyType,  
    this.selfAspect );
```

### ↑? alternativ

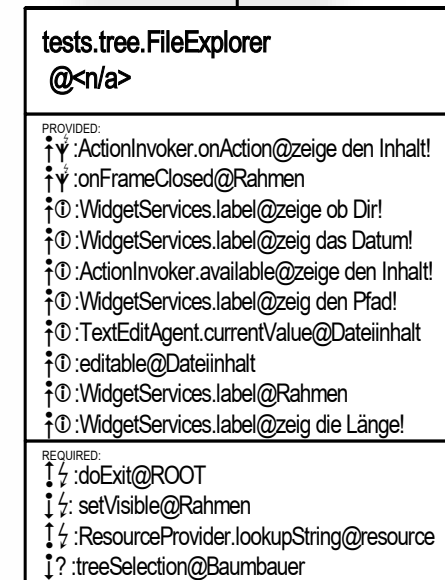
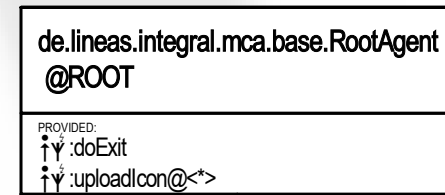
```
PropertyHandle valueOptHandle = addOptional (  
    MCAStandardService.StringPropertyType,  
    this.selfAspect );
```

## Benutzung

```
String value = valueHandle.get();  
valueHandle.set(inputString);
```

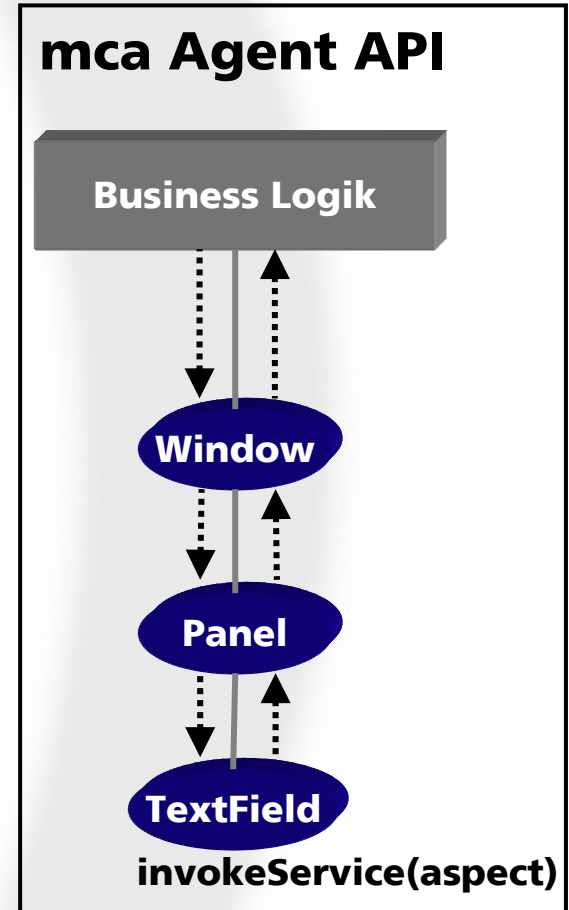
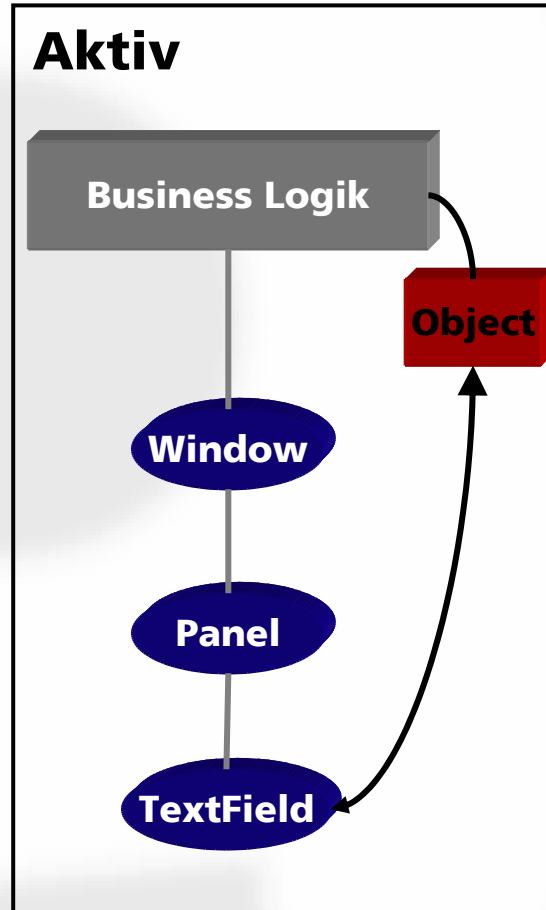
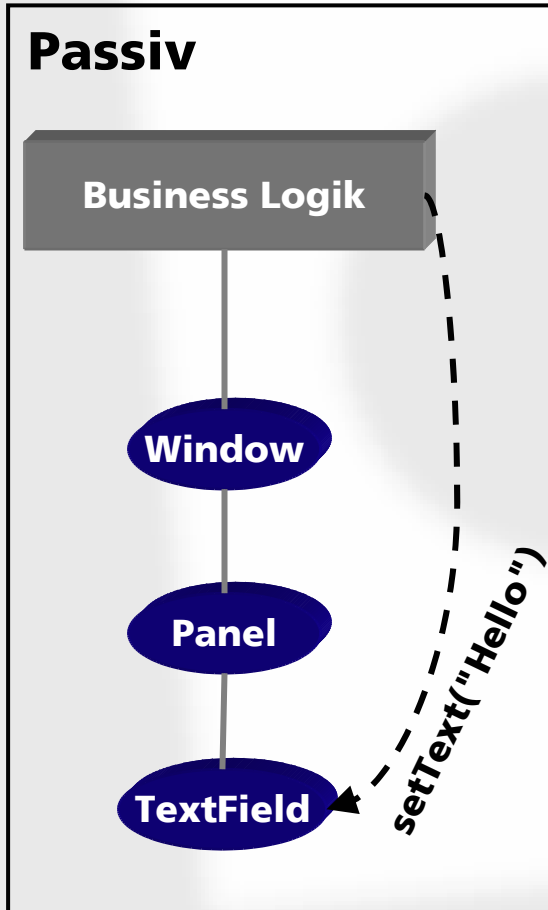
- ▶ Es gibt auch Downstream-Properties
- ▶ Tatsächlich ist Enable/Disable eine Downstream-Property des Buttons

- ▶ Visualisierung des aktuellen Baums wichtig für
  - ▶ Debugging
  - ▶ Einarbeitung
  - ▶ Wartungsarbeiten
- ▶ Jeder Knoten visualisiert einen Agenten mit
  - ▶ Klassennamen
  - ▶ Aspekt
  - ▶ Services und Properties
    - ▶ provided
    - ▶ required
    - ▶ optional
- ▶ Auf Knopfdruck Export nach MS Visio



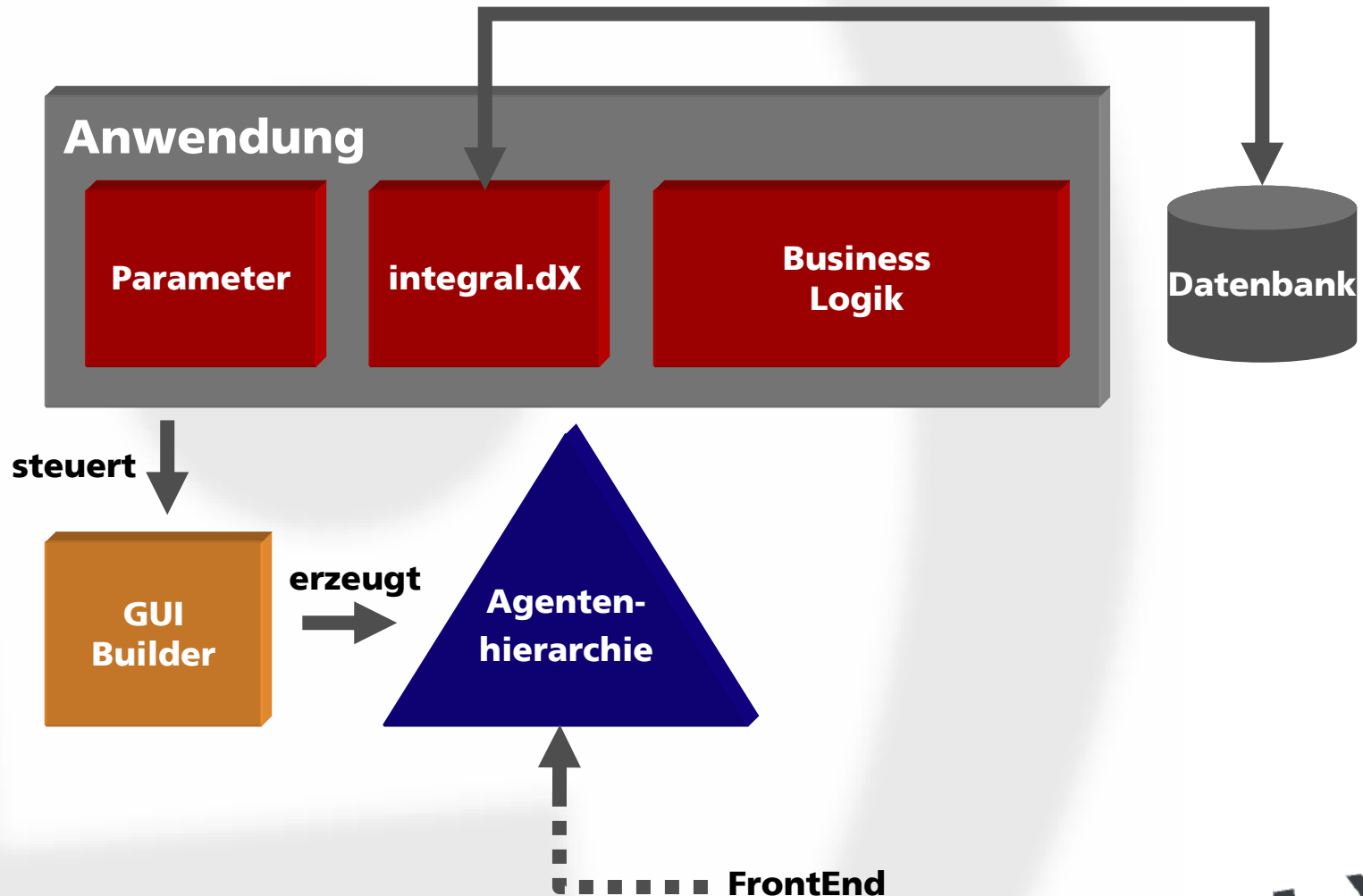
- ▶ Die integral.mca API ist eine Abwandlung des PAC-Patterns!
- ▶ GUI-Komponenten sind (PAC-)Agenten
  - ▶ keine Autonomie, keine Parallelität
- ▶ starke Kapselung
  - ▶ Kommunikation ausschließlich über Services und Properties
  - ▶ Agenten kennen einander nicht
- ▶ hierarchischer Aufbau
  - ▶ passend zum hierarchischen Aufbau der GUI
- ▶ standardisierte Kommunikation
  - ▶ entlang der Hierarchie
  - ▶ Services/Properties, upstream/downstream, required/optional
  - ▶ vereinfacht/spezialisiert gegenüber PAC
- ▶ interne Struktur der Agenten an PAC angelehnt, aber letztlich irrelevant (Hoher Zusammenhalt!)





# Konsequenzen

## Metadaten und mca – Integration mit dX



- ▶ Kontext
  - ▶ integral.dX
  - ▶ integral.dX GUI
- ▶ integral.mca
  - ▶ Anforderungen
  - ▶ Architektur
- ▶ APIs für GUIs
  - ▶ Model-View-Controller (MVC)
  - ▶ Presentation-Abstraction-Control (PAC)
- ▶ integral.mca API
  - ▶ Services
  - ▶ Properties
- ▶ Anwendungsbeispiel: Calculator
- ▶ Konsequenzen

# Anwendungsbeispiel: Calculator

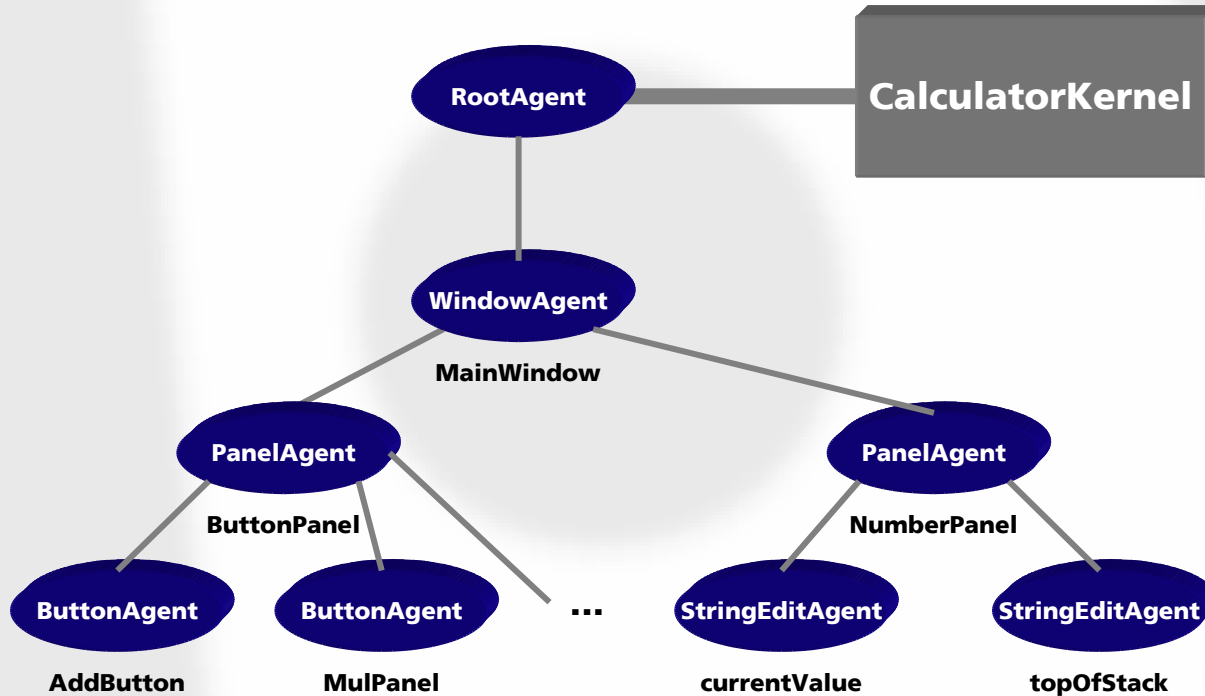
## Anforderungen

- ▶ 4 Grundrechenarten
- ▶ Eingabefeld
- ▶ Ergebnisfeld



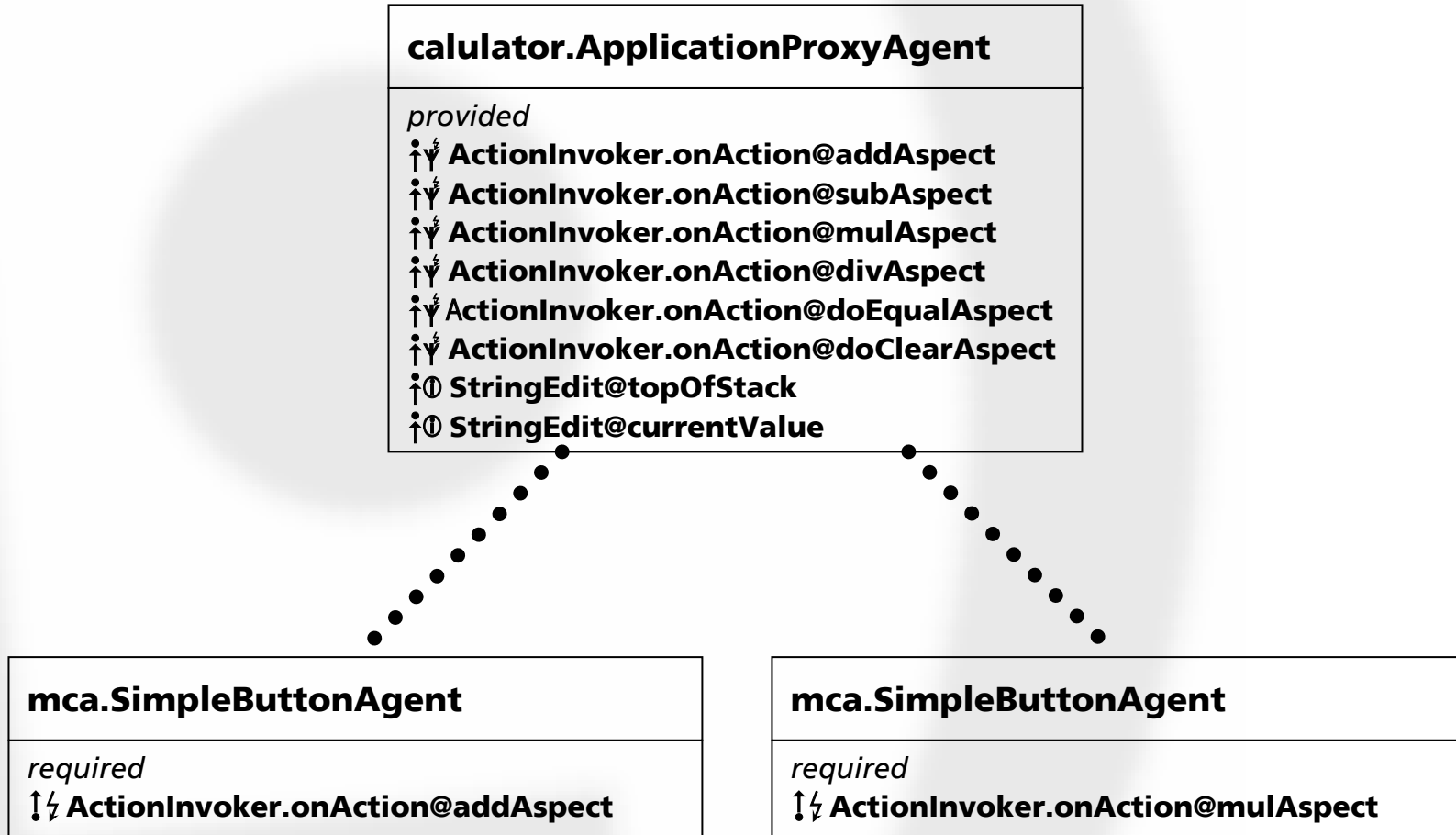
# Anwendungsbeispiel: Calculator

## Agentenhierarchie



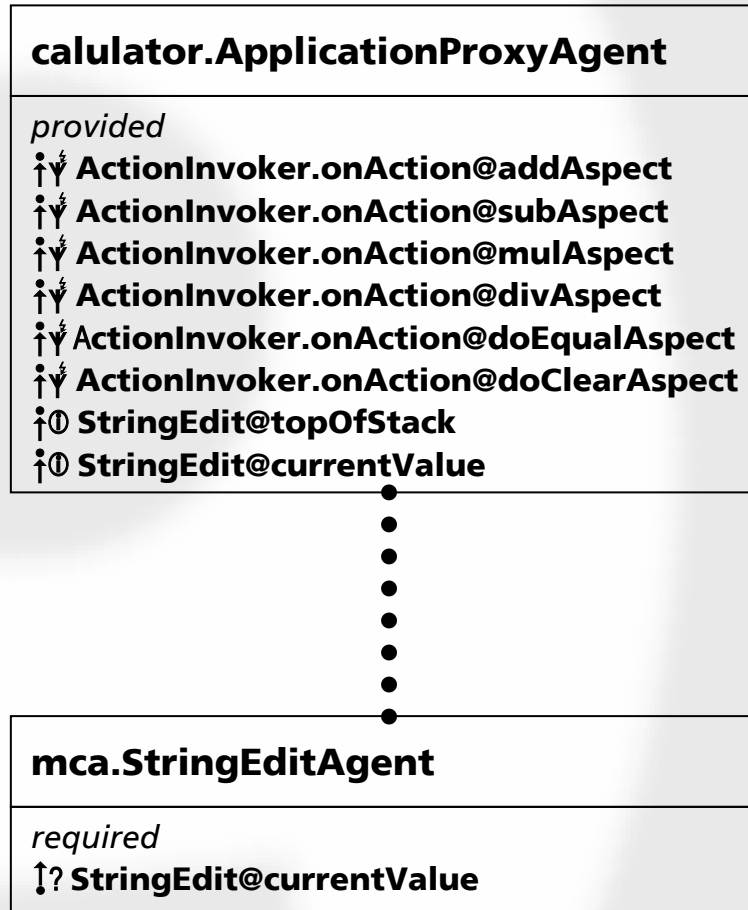
# Anwendungsbeispiel: Calculator

## Ausschnitt: Buttons



# Anwendungsbeispiel: Calculator

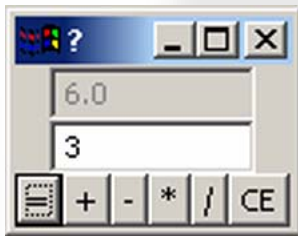
## Ausschnitt: currentValue



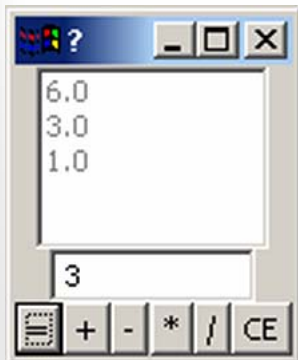
## Anwendungsbeispiel: Calculator

### Änderung der Anforderungen

- ▶ Kunde möchte die Anzeige wie bei einem Taschenrechner mit Drucker haben
- ▶ statt so

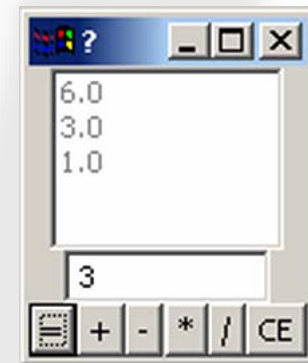
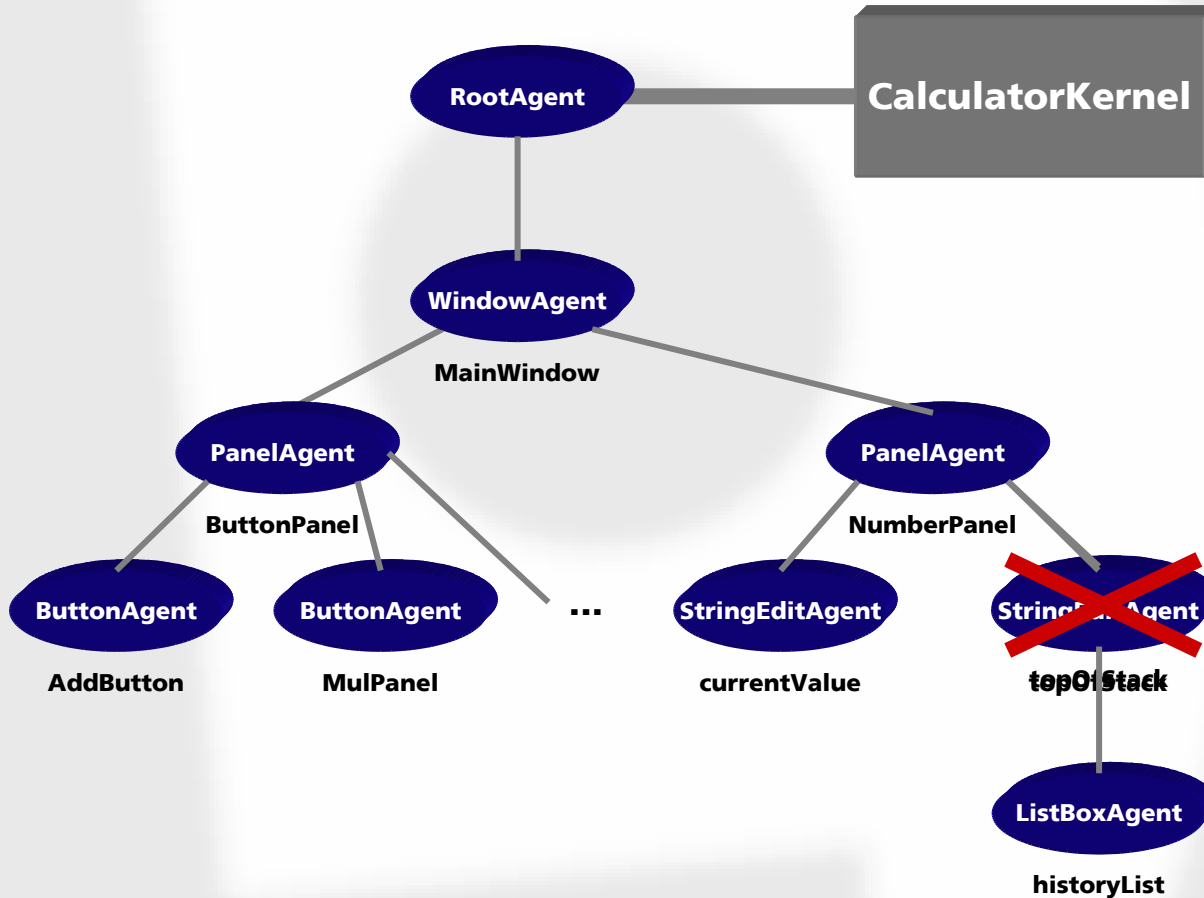


- ▶ soll die Maske nun so aussehen:



# Anwendungsbeispiel: Calculator

## Änderung der Anforderungen



- ▶ Kontext
  - ▶ integral.dX
  - ▶ integral.dX GUI
- ▶ integral.mca
  - ▶ Anforderungen
  - ▶ Architektur
- ▶ APIs für GUIs
  - ▶ Model-View-Controller (MVC)
  - ▶ Presentation-Abstraction-Control (PAC)
- ▶ integral.mca API
  - ▶ Services
  - ▶ Properties
- ▶ Anwendungsbeispiel: Calculator
- ▶ Konsequenzen

# Konsequenzen

## Erfahrungen (1)

- ▶ Neues Programmierparadigma
- ▶ Design Patterns haben sich als hilfreich herausgestellt
  - ▶ Auslösendes Elternteil / Auslösendes Kind
  - ▶ Interface
  - ▶ Vermittler (Kurier, Proxy, Multiplexer, Maskierer)
  - ▶ Perlenkette
  - ▶ Aspektfreie Eigenschaft
  - ▶ Interceptor / Adaptor / Converter
  - ▶ Komponenten
- ▶ Verschiedene Programmierregeln aus der praktischen Arbeit
  - ▶ Keine Applikationslogik in Agenten
  - ▶ Architektur der Business Logik für Einbindung vorbereiten
  - ▶ ...



# Konsequenzen

## Erfahrungen (2)

- ▶ Lose Kopplung der Agenten
- ▶ GUI änderbar/umstrukturierbar ohne Anwendung zu tangieren
- ▶ Anwendung ohne Anpassung der GUI änderbar
- ▶ Komponenten leicht baubar
- ▶ Generischer/automatischer GUI-Aufbau sehr gut möglich
  
- ▶ Einarbeitungsaufwand hoch
- ▶ Vorteile zeigen sich bei größeren Anwendungen

- ▶ integral.mca: Softwareentwicklungswerkzeug für verteilte grafische Benutzungsoberflächen
- ▶ integral.mca API: Abwandlung des PAC-Patterns
  - ▶ hierarchische Agentstruktur
  - ▶ standardisierter Kommunikationsmechanismus mit Services und Properties
- ▶ Lose Kopplung
  - ▶ Agenten und Anwendung kennen einander nicht
  - ▶ anonymisierte Kommunikation
- ▶ Vorteile
  - ▶ übersichtliche Objektstruktur
  - ▶ sehr hohe Flexibilität durch lose Kopplung
  - ▶ automatischer Aufbau von Oberflächen sehr leicht möglich
  - ▶ Hohe Wartbarkeit



**Vielen Dank für Ihre  
Aufmerksamkeit!**

