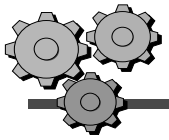


Werkzeugkonstruktion

Workplace Solutions

- ➔ Merkmale von Softwarewerkzeugen
- ➔ Trennung von Handhabung und Funktion: Interaktions- und Funktionskomponenten
- ➔ Lose Kopplung von Komponenten: das Beobachter-Muster

APCON Workplace Solutions

Bearbeitung von Konten aus anwendungsbezogener Sicht - Überweisen -

Workplace Solutions

Merkmale von Software-Werkzeugen

Workplace Solutions

Software-Werkzeuge

- haben einen **Namen**, ggf. ein **graphisches Symbol,**
- zeigen eine **Materialsicht**, sind aber auch **selbst im Blick**,
- bieten mögliche Handhabungen, d.h. **Operationen** an, um das Material unterschiedlich zu bearbeiten,
- zeigen jederzeit ihre **Einstellung**, die die Materialsicht bestimmt,
- sollen **keine Arbeitsabläufe implementieren**.

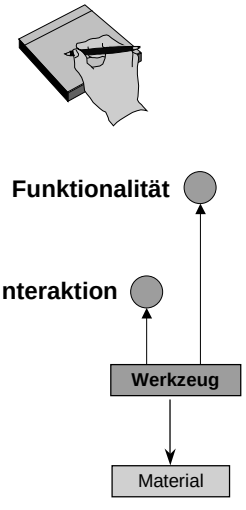
Aufgaben von Software-Werkzeugen

Workplace Solutions

Bei Software-Werkzeugen unterscheiden wir logisch zwischen

- der **Interaktion**: sie realisiert Handhabung und Präsentation des Software-Werkzeugs,
- der **Funktionalität**: sie ist der bewirkende und sondierende Teil des Software-Werkzeugs,
- der **Werkzeugrepräsentation**: sie repräsentiert ein Werkzeug von außen betrachtet und dient dem Zusammenbau von Werkzeugen aus Werkzeugkomponenten.

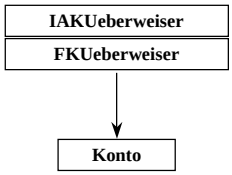
- Wird z.B. Java/Swing verwendet, so kann Interaktion direkt ein JFrame sein oder ein JFrame benutzen.
- Im Prinzip können die einzelnen Schnittstellen von beliebigen Klassen erfüllt werden.
- Bei monolithischen Werkzeugen existiert nur eine Fensterunterklasse, die alle Schnittstellen erfüllt.



Beispiel für die Schnittstelle einer Funktionskomponente

Workplace Solutions

```
public class FKUeberweiser {

    public void setzeAuftraggeber( int ktoNr, int blz ) {...}
        -- setzt das Konto des Auftraggebers

    public void setzeEmpfaenger( int ktoNr, int blz ) {...}
        -- setzt das Konto des Empfaengers

    public void ueberweisen( float betrag )
        -- überweist 'betrag' vom 'AuftraggeberKonto' auf
        -- das 'EmpfaengerKonto'

    public float empfaengerSaldo() {...}
        -- liefert den aktuellen Kontostand des Empfaengers

    public float auftraggeberSaldo {...}
        -- liefert den aktuellen Kontostand des Auftraggebers

    public boolean gueltigesKonto( int ktonr, int blz) {...}
        -- prüft, ob ein Konto mit 'ktonr' bei der Bank
        -- 'blz'existiert

    public boolean istUeberweisungMoeglich() {...}
        -- prüft, ob die Konten von Auftraggeber und
        -- Empfaenger bereits bestimmt worden sind

    public boolean istAuftraggeberSaldoVeraendert () {...}
        -- liefert 'true', wenn das Saldo des Auftraggebers
        -- verändert wurde

    public boolean istEmpfaengerSaldoVeraendert () {...}
        -- liefert 'true', wenn das Saldo des Empfängers
        -- verändert wurde

}
```



Zur Konstruktion einer Funktionskomponente

Workplace Solutions

Die **Funktionskomponente** (FK) bietet an ihrer Schnittstelle:

- Prozeduren, die eine Veränderung des FK-Zustands (`setzeAuftraggeber`, `setzeEmpfaenger`) bzw. des Material-Zustandes bewirken (`ueberweisen`),
- sondierende Funktionen, mit denen Informationen der FK bzw. des Materials erfragt werden können (`empfaengerSaldo, auftraggeberSaldo`),
- überprüfende Funktionen (`istEmpfaengerSaldoVeraendert, istAuftraggeberSaldoVeraendert`), mit denen u.a. Parameterwerte (`gueltigesKonto`) und Reihenfolgebedingungen (`istUeberweisungMoeglich`) getestet werden können.

Die Konstruktion einer FK ist unabhängig von den interaktionsspezifischen Teilen des Werkzeugs.

Zur Konstruktion einer Interaktionskomponente

Workplace Solutions

Die **Interaktionskomponente** (IAK)

- realisiert Handhabung und Präsentation eines Software-Werkzeugs,
- nimmt einen Strom von Systemereignissen entgegen, die durch Aktionen des Benutzers ausgelöst werden, und setzt sie in Programmereignisse um,
- setzt präsentationsbezogene Ereignisse (z.B. neue Farbeinstellungen) selbst um und leitet anwendungsbezogene Ereignisse an ihre FK weiter,
- abstrahiert mit Hilfe von Ein- und Ausgabekomponenten (z.B. Knöpfe, Menüs) vom zugrundeliegenden Fenstersystem, die sie für Darstellung und Benutzereingabe ruft.

Zusammenspiel von Funktions- und Interaktionskomponente

Workplace Solutions

Nach der Werkzeug-Metapher soll gelten:

- Jede Reaktion des Werkzeugs wird durch eine Aktion des Benutzers ausgelöst.

Daraus folgt:

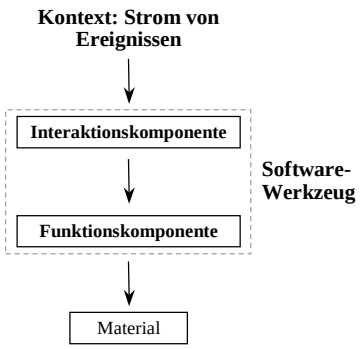
- die IAK benutzt die FK
- die FK benutzt das Material

Weiter gilt:

- Werkzeuge liefern ständige Rückkopplung

Daraus folgt:

- Wir haben ein Rückkopplungsproblem!



Lösungsansätze für das Rückkopplungsproblem (1)

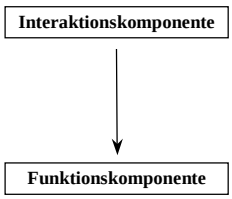
Workplace Solutions

Die IAK kennt die Effekte von Prozeduraufrufen und ruft geeignete sondierende Funktionen, um einen veränderten FK- bzw. Material darzustellen.

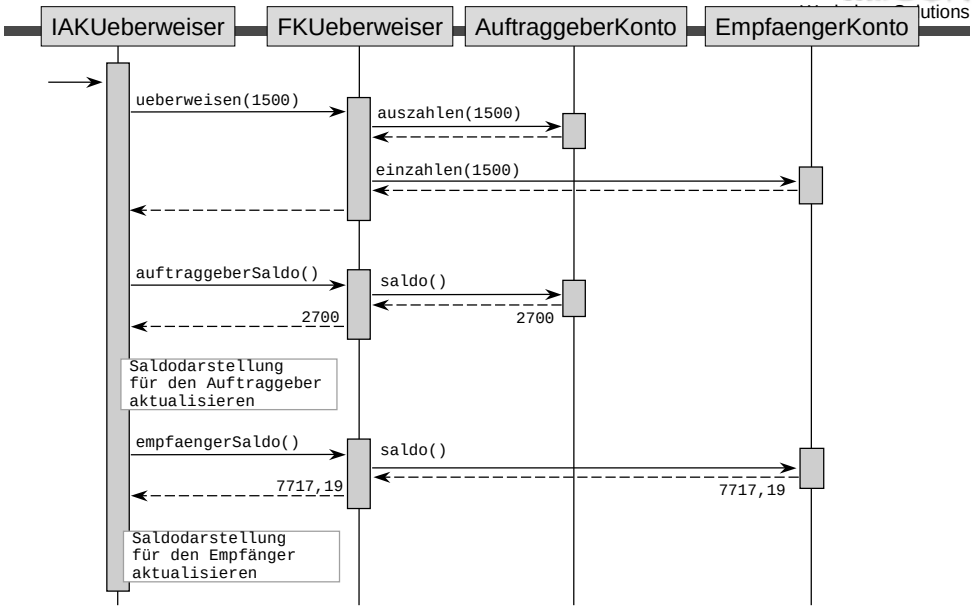
Das Geheimnisprinzip ist verletzt, da die IAK Kenntnis über die Umsetzung von Prozeduraufrufen besitzt !

Die IAK ruft die FK,

- um handhabungsbezogene Ereignisse in anwendungsbezogene Ereignisse umzusetzen,
- um bekannte Veränderungen des FK- bzw. Materialzustandes direkt abzufragen.



Lösungsansätze für das Rückkopplungsproblem (1)



Lösungsansätze für das Rückkopplungsproblem (2)

Workplace Solutions

Am Ende eines Prozeduraufrufs ruft die FK ihrerseits die IAK, um einen veränderten FK- bzw. Materialzustand aktualisiert darstellen zu lassen.

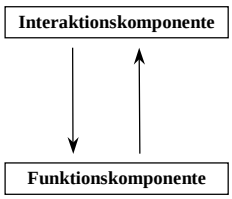
Die Aufgabentrennung von FK und IAK ist verletzt, da die FK über die Realisierung der Darstellung Kenntnis besitzt !

Die IAK ruft die FK,

- um handhabungsbezogene Ereignisse in anwendungsbezogene Ereignisse umzusetzen.

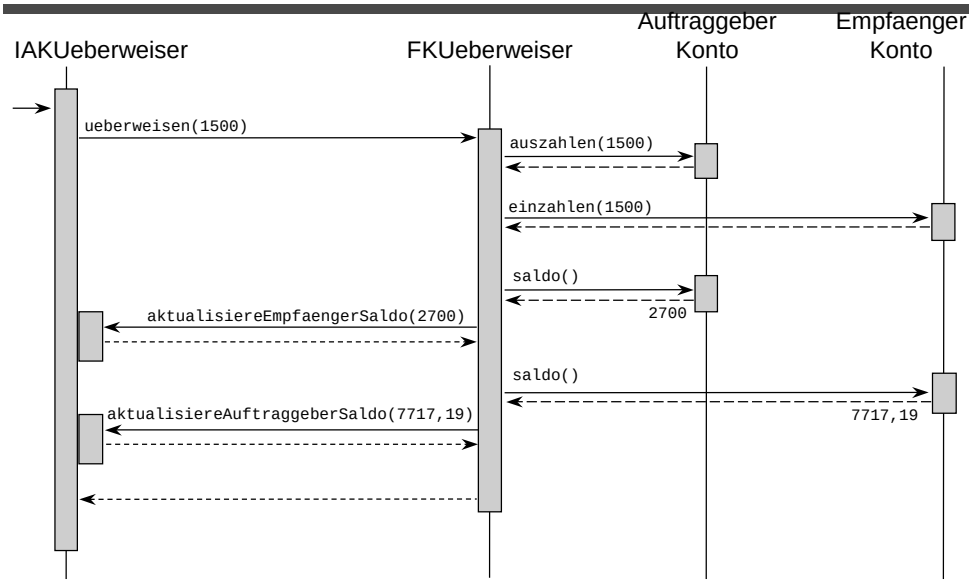
Die FK ruft die IAK,

- um eine aktualisierte Darstellung des FK- bzw. Materialzustandes an der Oberfläche zu veranlassen.



Lösungsansätze für das Rückkopplungsproblem (2)

Workplace Solutions



Lösungsansatz für das Rückkopplungsproblem (3)

Workplace Solutions

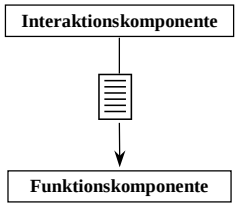
Nach jedem Systemereignis ruft die IAK die überprüfenden Funktionen der FK, um einen veränderten FK- bzw. Materialzustand unmittelbar nachzuvollziehen. Wir bezeichnen diesen Lösungsansatz als Polling-Ansatz.

Die IAK ist fast ausschließlich mit Polling beschäftigt !

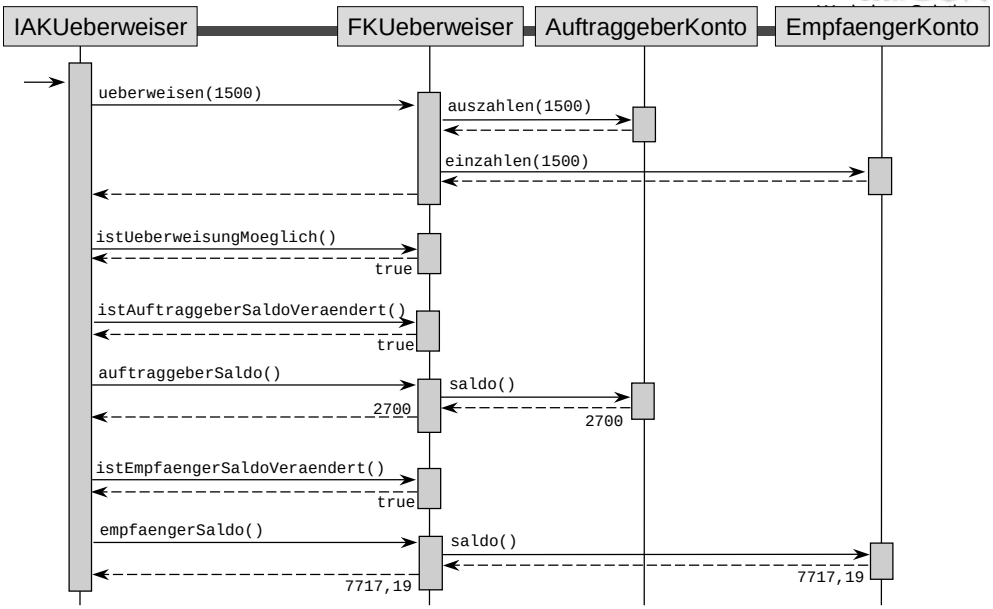
Softwaretechnisch ist Polling die sauberste der drei Lösungen und vermeidet die Nachteile der ersten beiden Lösungen.

Die IAK ruft die FK,

- um handhabungsbezogene Ereignisse in anwendungsbezogene Ereignisse umzusetzen,
- um sich über mögliche Veränderungen des FK- bzw. Materialzustandes zu informieren.
- um die veränderten Informationen zu erfragen.



Lösungsansätze für das Rückkopplungsproblem (3)

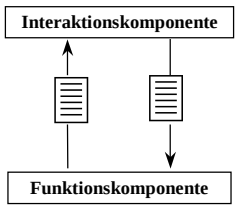


Weiterentwicklung des Polling-Ansatzes: Der benachrichtigte Beobachter

Workplace Solutions

Konzept:

- Die FK signalisiert Zustandsänderungen (des Materials und ihres eigenen Zustands).
- Die IAK beachtet diese Signale und ruft entsprechende sondierende Funktionen.
- Folge: Die IAK muß in der FK bekannt sein.



Aufrufe,Ereignisse, Signale

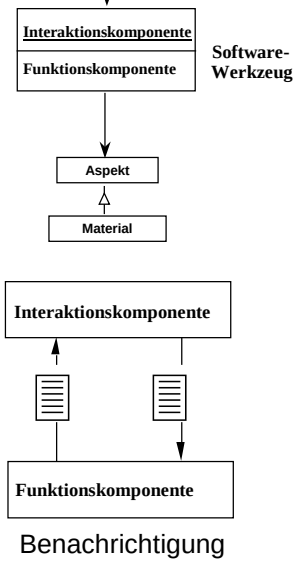
Kontext: Strom von Ereignissen

Workplace Solutions

Wir unterscheiden:

- Beim **Aufruf** (Call) kennt der Rufer (Klient) den Gerufenen (Anbieter) und erwartet eine bestimmte Dienstleistung.
- Ein **Ereignis** (Event) wird vom Erzeuger oder Verteiler an einen bestimmten Empfänger weitergeleitet. Eine Reaktion des Empfängers wird nicht erwartet.
- Ein **Signal** (Broadcast) wird vom Erzeuger an die Umgebung ausgesandt. Prinzipiell sind die Empfänger des Signals anonym. Die Reaktion auf ein Signal ist meist ein Aufruf des Erzeugers.

Da uns die gängigen oo Sprachen keinen eigenen Ereignis- oder Signalmechanismus anbieten, konstruieren wir die Benachrichtigung mit Hilfe des Aufrufmechanismus.



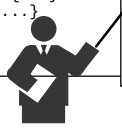
Benachrichtigung

Struktur des Beobachter-Mechanismus: ein Zwischenergebnis

Workplace Solutions

```
public class IAKUeberweiser {

  public void beachteAenderung()
  {
    if _funktion.istUeberweisungMoeglich()...
    if _funktion.istAuftraggeberSaldoVeraendert()...
    if _funktion.istEmpfaengerSaldoVeraendert()...
  }
  ...
  private FKUeberweiser _funktion;

} -- class IAKUeberweiser
```

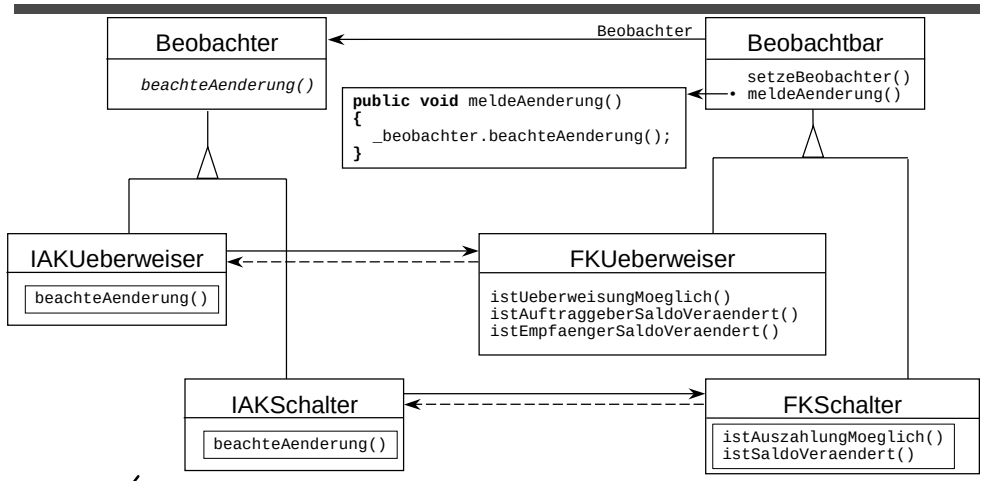
```
public class FKUeberweiser {

  public void setzeIAK(IAK i)
  { _interaktion = i; }

  public void meldeAenderung()
  {
    _interaktion.beachteAenderung();
  }

  public void ueberweisen( float betrag )
  {
    _auftraggeberKonto.auszahlen( betrag );
    _empfaengerKonto.einzahlen( betrag );
    meldeAenderung();
  end;

  public boolean istUeberweisungMoeglich() {...}
  public boolean istAuftraggeberSaldoVeraendert () {...}
  public boolean istEmpfaengerSaldoVeraendert () {...}
  ...
  private IAKUeberweiser _interaktion;
} -- class FKUeberweiser
```



Entwurfsentscheidung: Die FK benutzt genau eine Operation an der Schnittstelle der IAK.

Generalisierung des Beobachter-Mechanismus: Die Klassen **Beobachter** und **Beobachtbar**

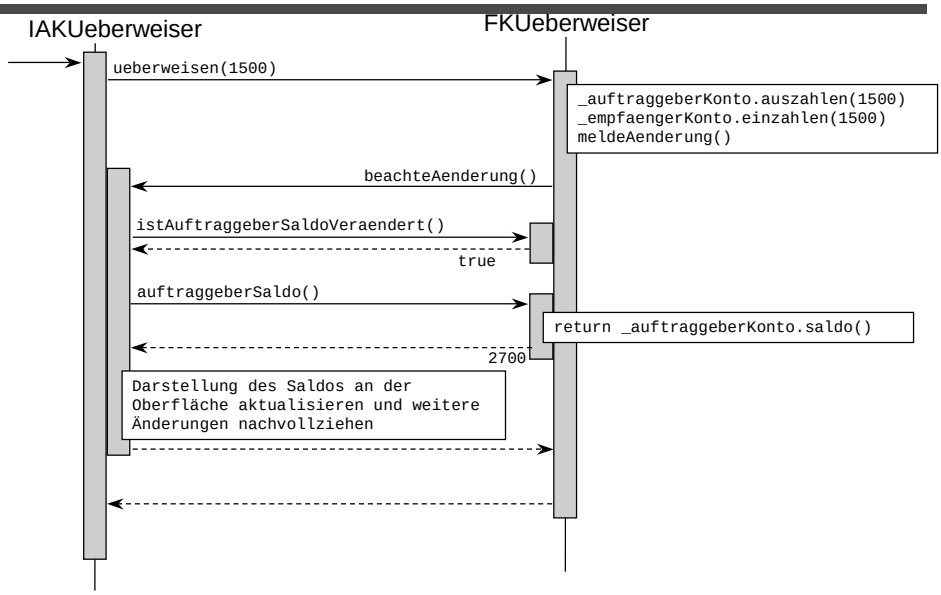
Workplace Solutions



Eine IAK kennt die ihr zugeordnete FK unter der vollen Schnittstelle.
Die FK benachrichtigt ihre IAK unter ihrer abstrakten Schnittstelle Beobachter.

Dynamik des Beobachter-Musters

Workplace Solutions

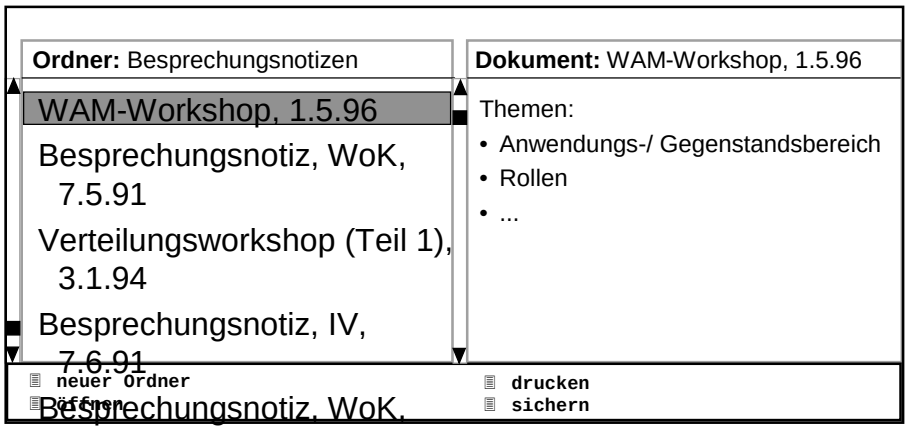


Vom Software-Werkzeug zu Werkzeughierarchien

Workplace Solutions

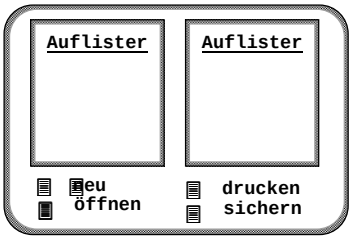
Beispiel: Ein Werkzeug für Ordnerinhalte

Workplace Solutions



Problem: Komplexe Handhabung und Materialzusammenhang

Workplace Solutions

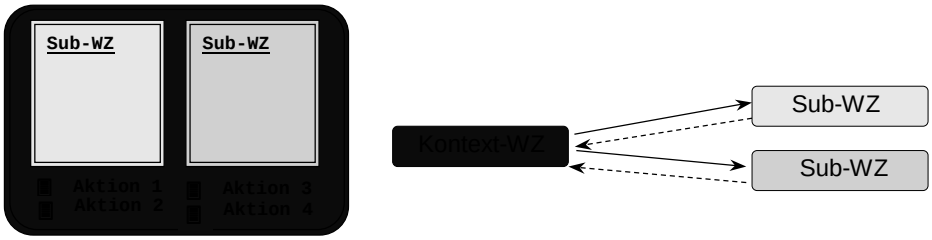


Eine schlechte Lösung wäre,

- eine komplexe Funktionskomponente zu entwerfen, die die gesamte Handhabung realisiert.
- die gleiche Interaktionsform (Auflisten von Inhalten) doppelt zu implementieren.
- Die Materialien sind miteinander verknüpft.

Kontext- und Subwerkzeuge

Workplace Solutions



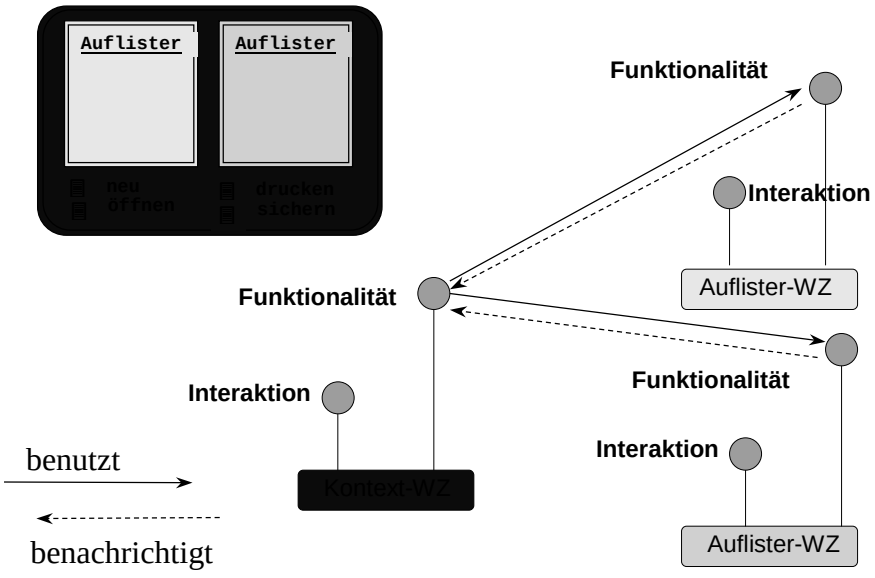
Zur Konstruktion komplexerer Werkzeuge verwenden wie Werkzeugkomponenten.

- Kleinere Einheiten lassen sich leichter entwerfen und pflegen.
- Die einzelnen Sub-Werkzeuge sind unabhängig voneinander.
- Das Kontextwerkzeug kann die Sub-Werkzeuge parametrisieren.
- Diese Maßnahmen erhöhen die Wiederverwendbarkeit!

Werkzeugkomposition

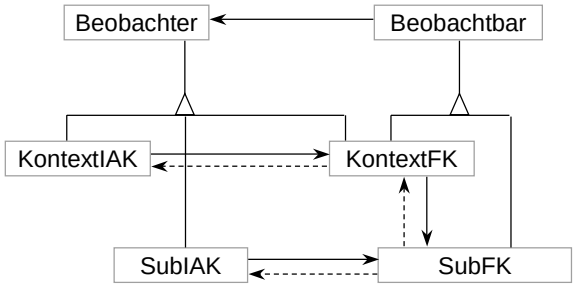
Beziehungen zu Sub-WZ

Workplace Solutions



Werkzeughierarchien mit FK-IAK

Workplace Solutions

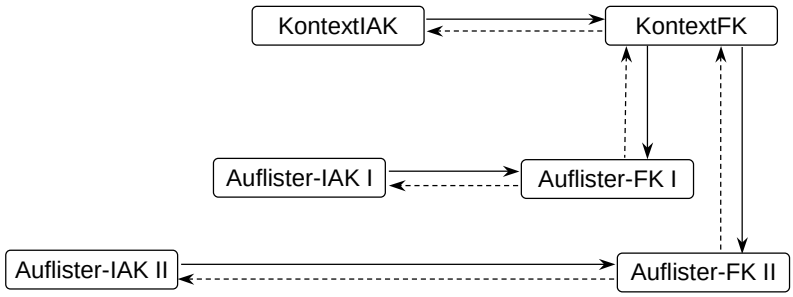


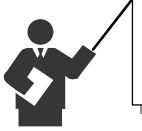
Werkzeuge als Komponenten

- Werkzeuge aus Bausteinen zu Werkzeughierarchien zusammenzusetzen,
- gleiche Bausteine durch Exemplare derselben Klasse zu realisieren.

Dynamik von Werkzeug und Sub-Werkzeugen

Workplace Solutions





Die Architektur bestimmt die Komponenten,

- die dynamisch erzeugt werden und
- die über den Beobachtermechanismus Ereignisse auslösen.

Anwendung des Beobachter-Mechanismus bei der Werkzeug-Subwerkzeug Konstruktion

Workplace Solutions

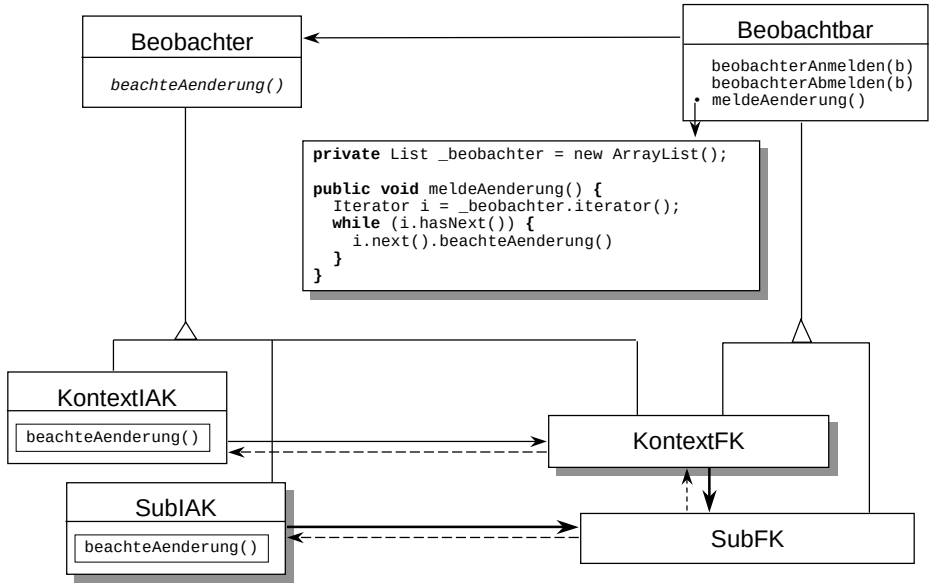
```
public class AuflisterFK {

  public void SetzeIAK (Beobachter I) {...};
  public void setzeKontextFK (Beobachter K) {...};

  public void meldeAenderung() {
    if (_interaktion != null)
      _interaktion.beachteAenderung();
    if (_kontextFK != null)
      _kontextFK.beachteAenderung();
      ...
    }


  private Beobachter _interaktion;
  private Beobachter _kontextFK;

} -- class AuflisterFK
```

Konzept:

- Eine Sub-FK signalisiert die Erledigung einer Teilaufgabe an ihre einbettende FK.
- Die Kontext-FK fragt nach dem Ergebnis der Teilaufgabe.
- Die Kontext-FK erledigt ggf. die noch offenen Teilaufgaben oder ruft weitere Sub-FKs bzw. Kontext-FKs.

Weitere Generalisierung des Beobachtermechanismus: mehrere Beobachter

Workplace Solutions



Weitere Anwendungen des Beobachter-Mechanismus

Workplace Solutions

Konto 174653845

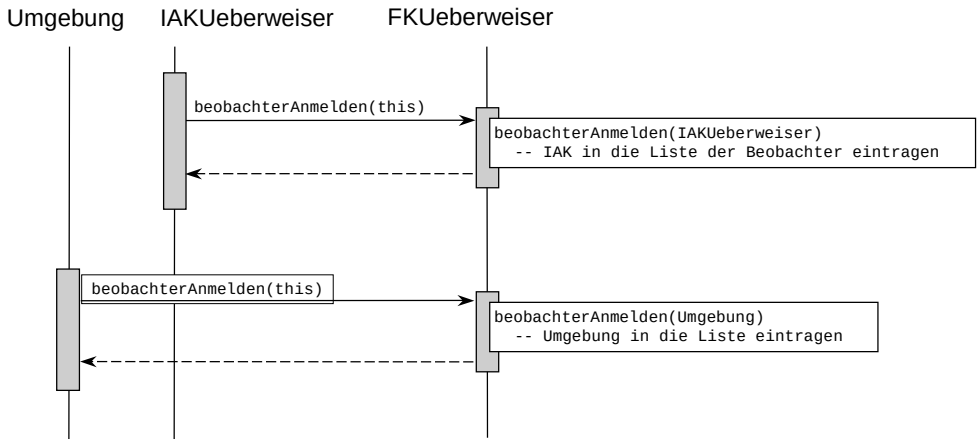
Der Beobachter-Mechanismus läßt sich darüber-hinaus sinnvoll anwenden, wenn

- innerhalb einer Umgebung zwei Werkzeuge **gleichzeitig auf demselben Material** arbeiten,
- einer Funktionskomponente zwei (oder mehr) Interaktionskomponenten zugeordnet sind

und jeweils eine aktuelle Sicht auf das Material präsentiert werden soll.

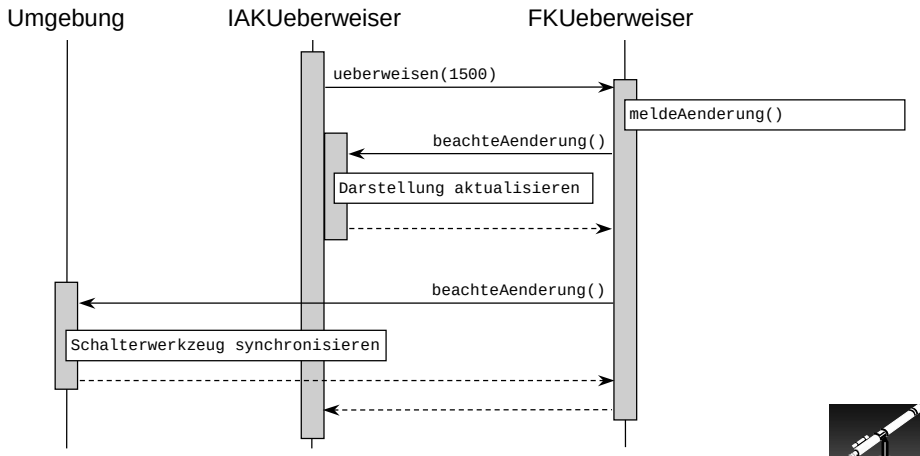
Dynamik des Beobachter-Mechanismus: Registrieren mehrerer Beobachter

Workplace Solutions

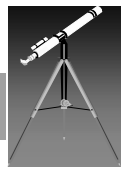


Dynamik des Beobachter-Mechanismus: Melden einer Änderung an mehrere Beobachter

Workplace Solutions



Der Beobachter-Mechanismus ist ein Beispiel für ein **Entwurfsmuster**!



Trennung von Interaktion und Funktion

Workplace Solutions

benachrichtigt

Die Benachrichtigung erfolgt bei über Events!

evtObject

+register()
+unregister()
+announce()
+owner()

«Schnittstelle»
EventSubject

+isAnnouncing()
+startAnnouncement()
+endAnnouncement()

Verschickt Events

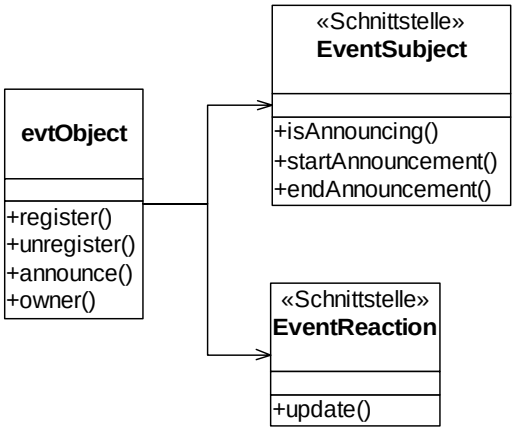
«Schnittstelle»
EventReaction

+update()

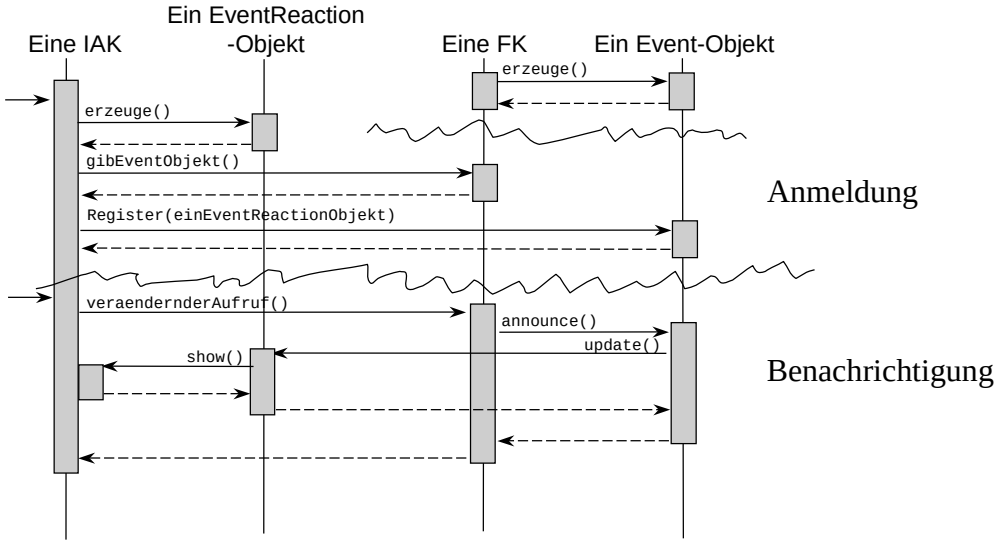
Reagiert auf Events

Ein Funktionsteil bietet an seiner Schnittstelle Events für fachliche Zustandsübergänge an.

Eventing



Sequenzdiagramm: Eventing



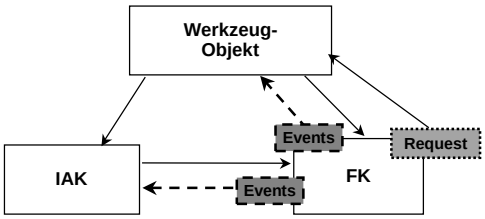
Eventing vs. Requesting

- Reaktionsmechanismen -

Workplace Solutions

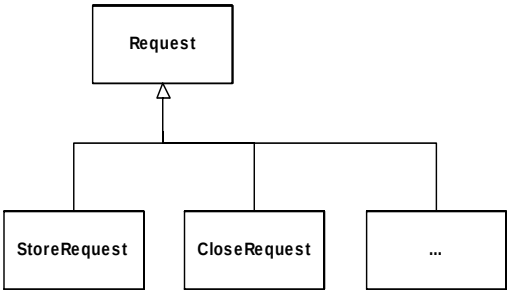
Events werden bei **Zustandsänderungen** von FK zu IAK oder von FK zu Kontext-FK verschickt (z.B. Änderung am Material).

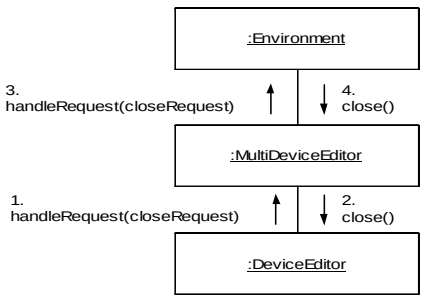
Requests werden bei **Anforderungen**, die eine FK oder ein Tool nicht erfüllen kann und sich daher auch nicht ihr/sein Zustand ändert, an den Kontext gesandt (z.B. Schließen).



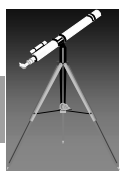
Requests

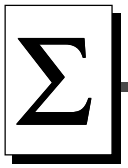
Workplace Solutions





Der Request-Mechanismus ist ein Beispiel für ein **Entwurfsmuster**!





Fazit

Workplace Solutions

- Bei der Konstruktion von Software-Werkzeugen werden **Handhabung** und **Funktion** in getrennten Komponenten modelliert.
- Die Klassen **Beobachter** und **Beobachtbar** erlauben ihre vollständige Entkopplung.
- Eine beliebige Anzahl von **Interaktionskomponenten** kann eine **Funktionskomponente** beobachten, d.h. mehrere Sichten auf ein Material sind möglich.
- Der Beobachter-Mechanismus kann darüberhinaus zur **losen Kopplung** verschiedener Werkzeuge verwendet werden, die gleichzeitig auf demselben Material arbeiten.