

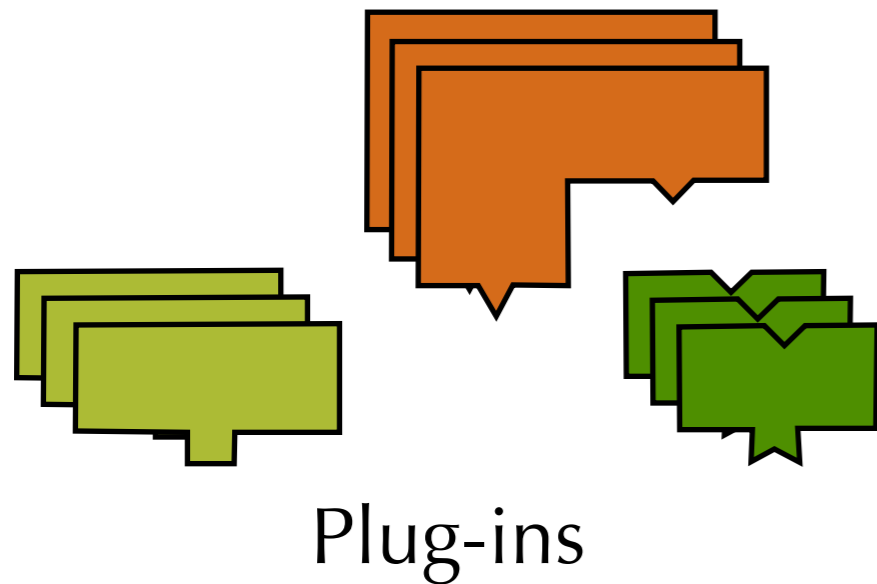
Integrationsmodelle für ein .NET Plug-in Framework

Reinhard Wolfinger, Herbert Prähofer
Christian Doppler Labor
für Automated Software Engineering
Johannes Kepler University, Linz, Austria
wolfinger@ase.jku.at, herbert.praehofer@ase.jku.at

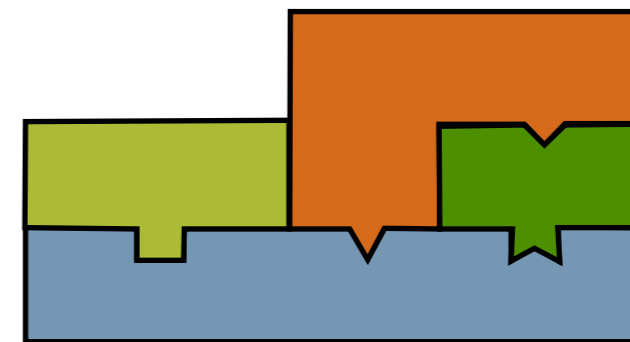
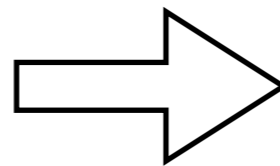
SE2007 - 30. März 2007

.NET Plug-in Framework

Plug-in Framework



Applikation mit Erweiterungen



Kernapplikation

Was ist ein Plug-in?

- einzeln installierbare Einheit,
- für den Benutzer sichtbares Feature
- integrierbar in größere Applikation

Warum Plug-ins?

Erweiterbarkeit

Erweiterungen durch Dritte

- Szenario: Upgrading
Nach Erstkauf Features dazukaufen

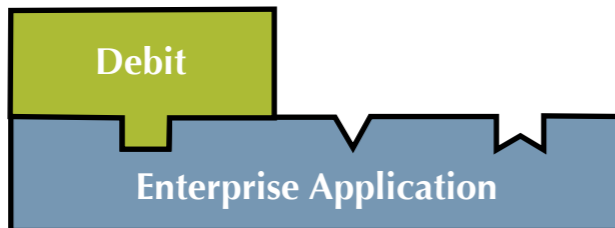
Individuelle Arbeitsumgebung

Arbeitsumgebung jedem Benutzer individuell anpassen

- Szenario: Verkaufsunterstützung
Live-Vorschau der Anwendung, "I know it when I see it"
- Szenario: Mietsoftware
Features aktivieren, Nutzungsdauer messen und verrechnen
- Szenario: Benutzerservice am Help Desk
Arbeitsumgebung am Help Desk nachstellen
- Szenario: Training
Arbeitsumgebung für aktuelle Trainingseinheit, Funktionalität schrittweise hinzufügen
- Szenario: Rollenspezifische Sichten
Arbeitsumgebung wechseln von Rolle "Buchhalter" zur Rolle "Controller" (ohne Neustart)

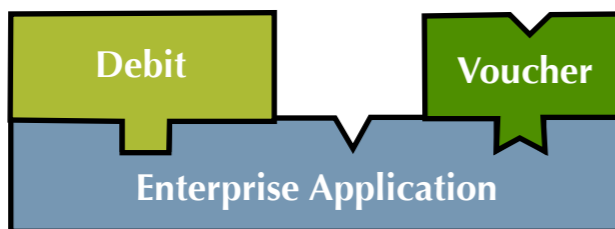
Szenario: Training

Lektion 1: Buchen einer Ausgangsrechnung



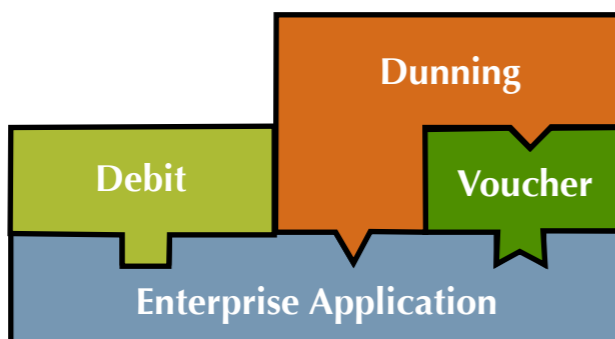
Trainee konzentriert sich auf Lerninhalt der Lektion 1.

Lektion 2: Erstellen einer Offene Posten - Liste



Trainee konzentriert sich auf Lerninhalt von Lektion 2 und integriert mit Wissen aus vorhergehender Lektion.

Lektion 3: Erstellen von Mahnschreiben

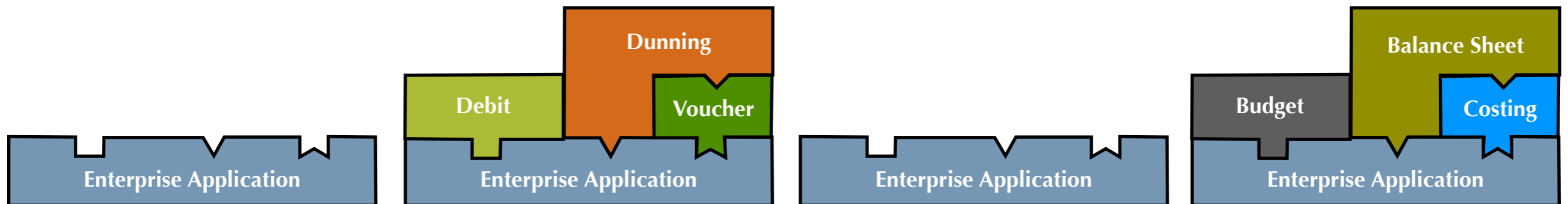
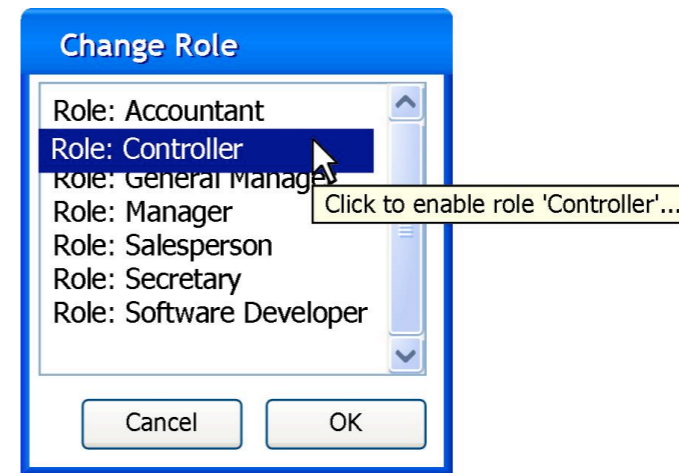
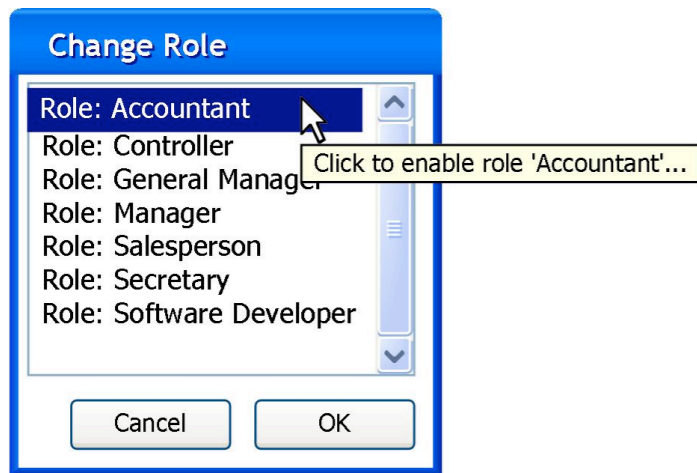


Das Anwendungsprogramm wird langsam aufgebaut und entwickelt sich so Schritt-für-Schritt mit dem Lernfortschritt des Trainees mit.

Szenario: Rollenwechsel

Eine Mitarbeiterin arbeitet vormittags in der Rolle "Buchhalter"...

...und am nachmittag wechselt sie in die Rolle des "Controllers".



Passen sie die Arbeitsumgebung mit einem Mausklick an ihre aktuelle Rolle an. Sie sehen nur die Funktionen die sie gerade benötigen.

Unterschiede Eclipse

Eclipse Development Platform

- führende Plug-in Plattform
- basiert auf Java

Unterschied 1: Technische Plattform Microsoft .NET

- Win32-Applikation des Firmenpartners soll integriert werden
komfortable Einbindung mit .NET Interop
- bringt neue technische Herausforderungen,
- aber auch neue Möglichkeiten bei Umsetzung

Unterschied 2: Spezielle Anforderungen

- Eclipse ist eine Development Tools Plattform
- spezielle zusätzliche Anforderungen bei Unternehmenssoftware
(Sicherheit, Zuverlässigkeit, Versionierung)

Spezielle Anforderungen

Sicherheit

beschränken was ein Plug-in darf

Zuverlässigkeit

den Host vor Abstürzen eines Plug-ins schützen

Versionierung

Host und Plug-in unabhängig versionieren

Eine Plug-in Architektur für Unternehmenssoftware soll Offenheit ermöglichen aber zugleich zuverlässig und sicher sein.

Plug-in Framework

1

Contracts

Erweiterungspunkte spezifizieren

2

Deployment

Erweiterung verpacken und installieren

3

Discovery

Installierte Erweiterungen entdecken

4

Qualification

Prüfen ob Erweiterung passt

5

Static Integration

Erweiterung statisch in Kernapplikation integrieren

6

Activation

Erweiterung laden und dynamisch integrieren

7

Deactivation

Erweiterung entladen und Ressourcen freigeben

Kontrakte spezifizieren

Slot

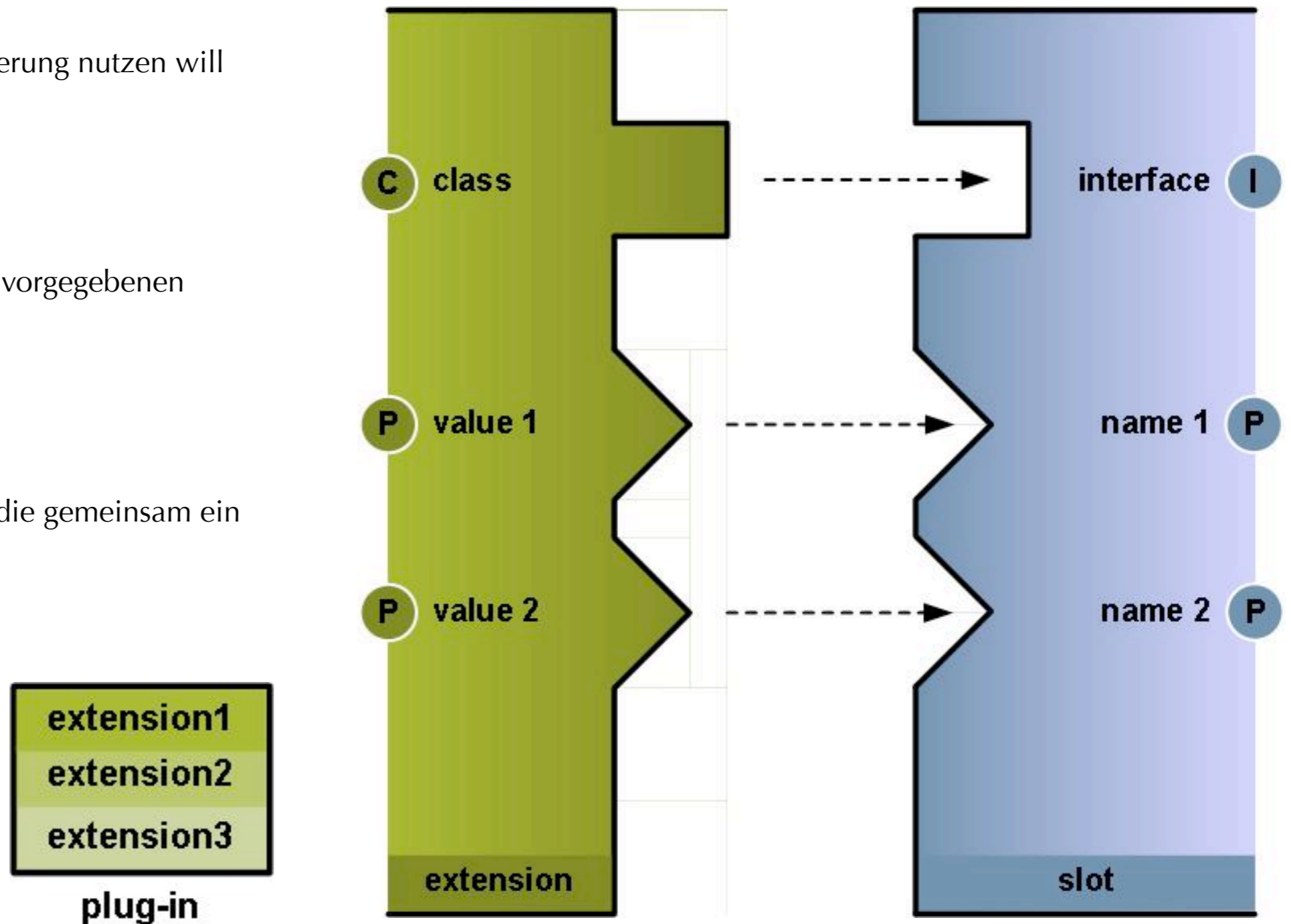
Spezifiziert wie Host eine Erweiterung nutzen will (Interface).

Extension

Implementierung des vom Slot vorgegebenen Objektmodells (Klasse).

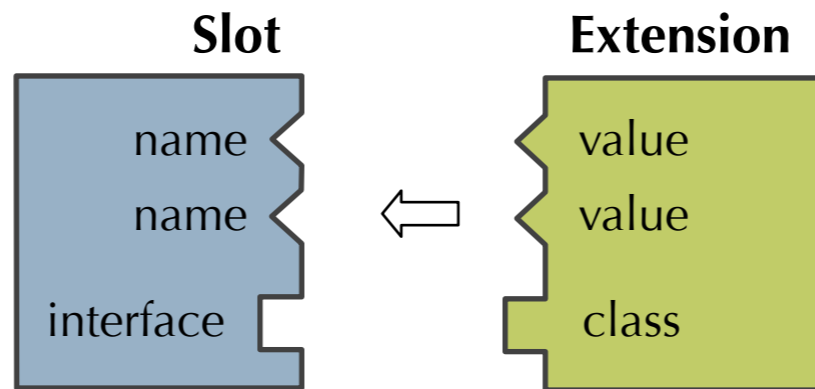
Plug-in

Sammlung von Erweiterungen die gemeinsam ein Feature realisieren.

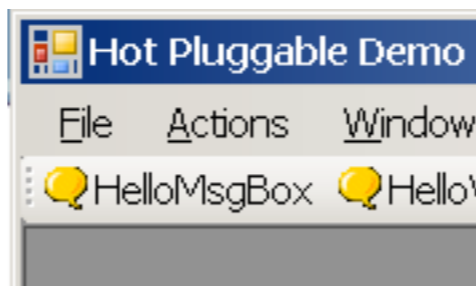


Kontrakt - Attribute

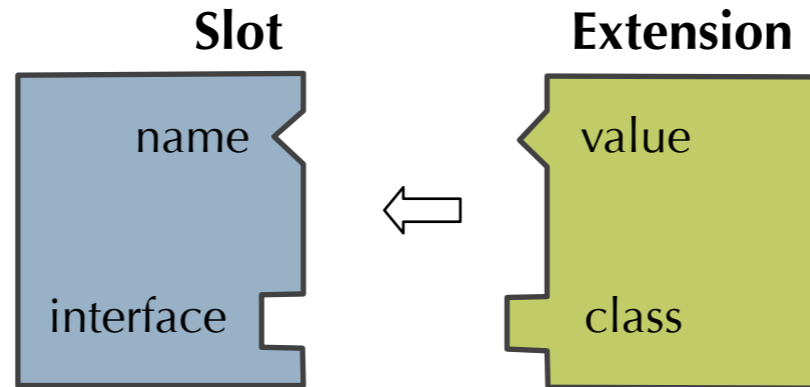
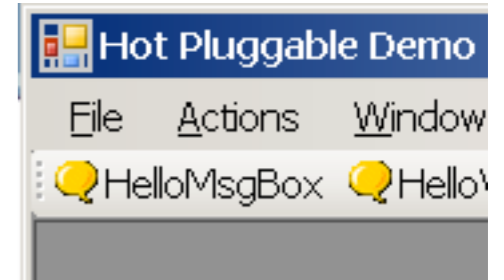
```
[Slot("MenuItem")]  
[Name("Text", typeof(string))]  
[Name("Icon", typeof(string))]  
interface IMenuItem {  
    ...  
}
```



```
[Extension(  
    Name= "PrintItem",  
    Slot="MenuItem")]  
[Value("Text", "Print")]  
[Value("Icon", "Print.ico")]  
class PrintItem: IMenuItem {  
    ...  
}
```



Kontrakt - benutzerdef. Attribute



```
[Slot("MenuItem")]
```

```
public class MenuItemAttribute : Attribute {  
    public string Text;  
    public string Icon;  
}
```

```
[Slot("MenuItem")]
```

```
[Property(typeof(MenuItemAttribute))]
```

```
interface IMenuItem {  
    ...  
}
```

```
[Extension(Name="PrintItem", Slot="MenuItem")]
```

```
[MenuItem(Text="Print", Icon="Print.ico")]
```

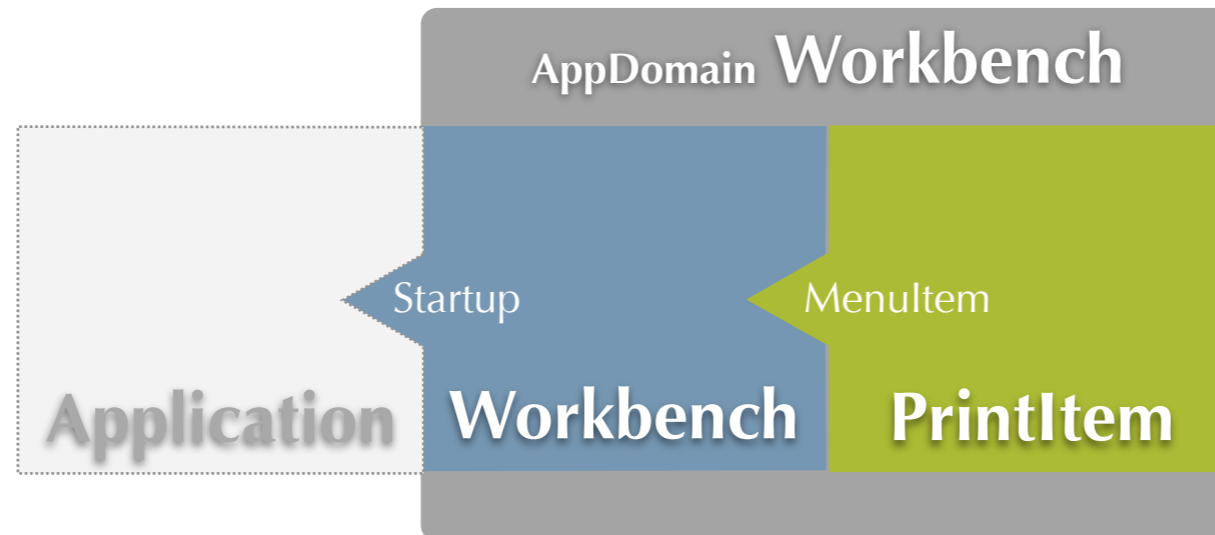
```
class PrintItem : IMenuItem {  
    ...  
}
```

Integrationsmodelle (1)

Tightly Coupled - no Isolation

Host und Plug-in teilen eine AppDomain

- für Komponenten der Kernapplikation



```
[Extension(Name="Workbench", Slot="Startup")]  
[Isolation(None)]  
[Domain("Workbench")]  
class Workbench : IStartup {  
    ...  
}
```

```
[Extension(Name="PrintItem", Slot="MenuItem")]  
[Domain("Workbench")]  
class PrintItem : IMenuItem {  
    ...  
}
```

Integrationsmodelle (2)

AppDomain Isolation

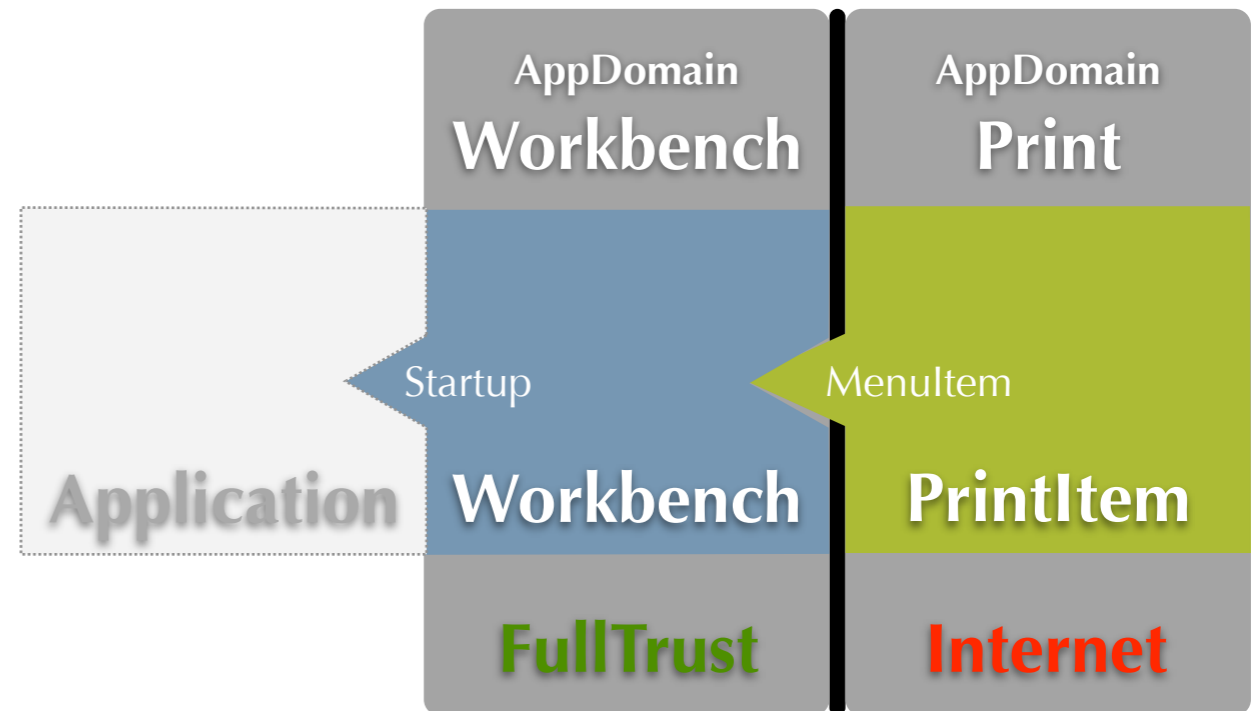
Plug-in in separater AppDomain aktivieren

- Plug-in entladen
- beschränken was Plug-in darf

```
[Slot("MenuItem")]  
[Security(PermissionSet=Internet)]  
interface IMenuItem {  
    ...  
}
```

```
[Extension(Name="Workbench", Slot="Startup")]  
[Isolation(AppDomain)]  
[Domain("Workbench")]  
class Workbench : IStartup {  
    ...  
}
```

```
[Extension(Name="PrintItem", Slot="MenuItem")]  
[Domain("Print")]  
class PrintItem : IMenuItem {  
    ...  
}
```

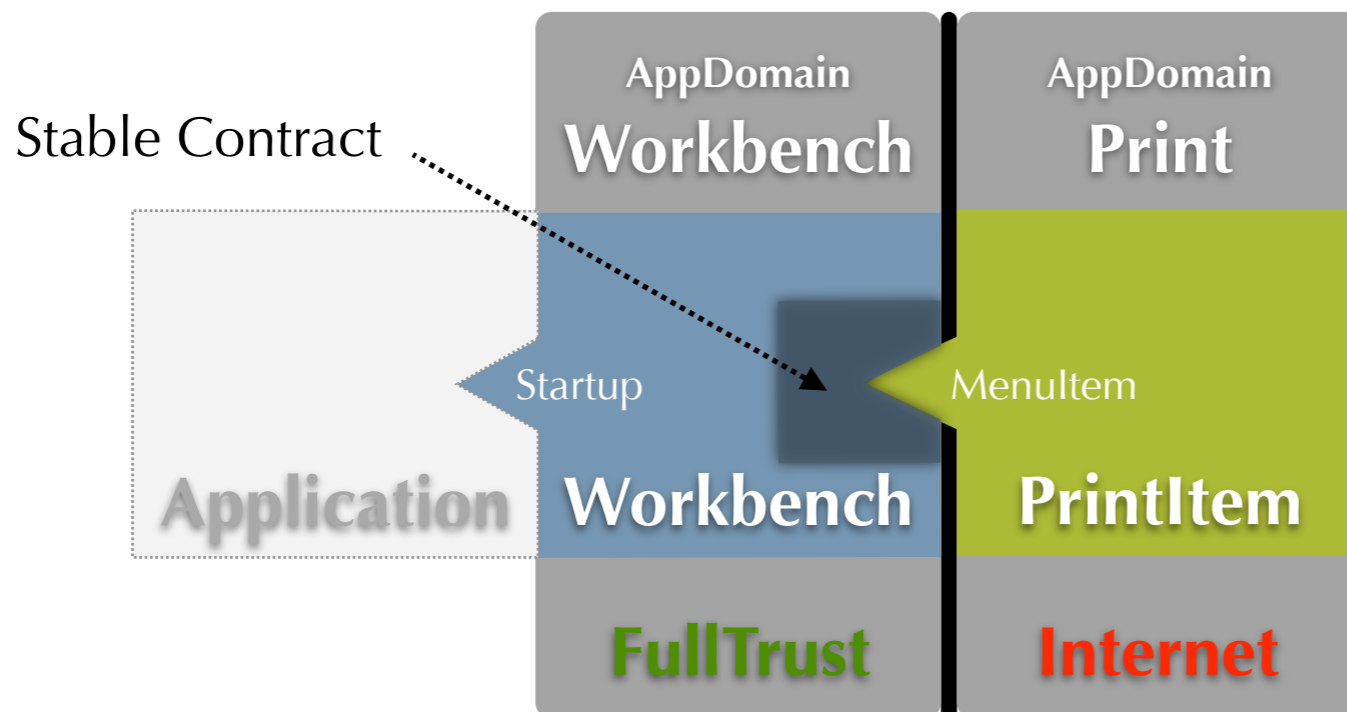


Integrationsmodelle (3)

Stable Contract

Stabiler Vertrag als Abgrenzung

- erlaubt Versionierung

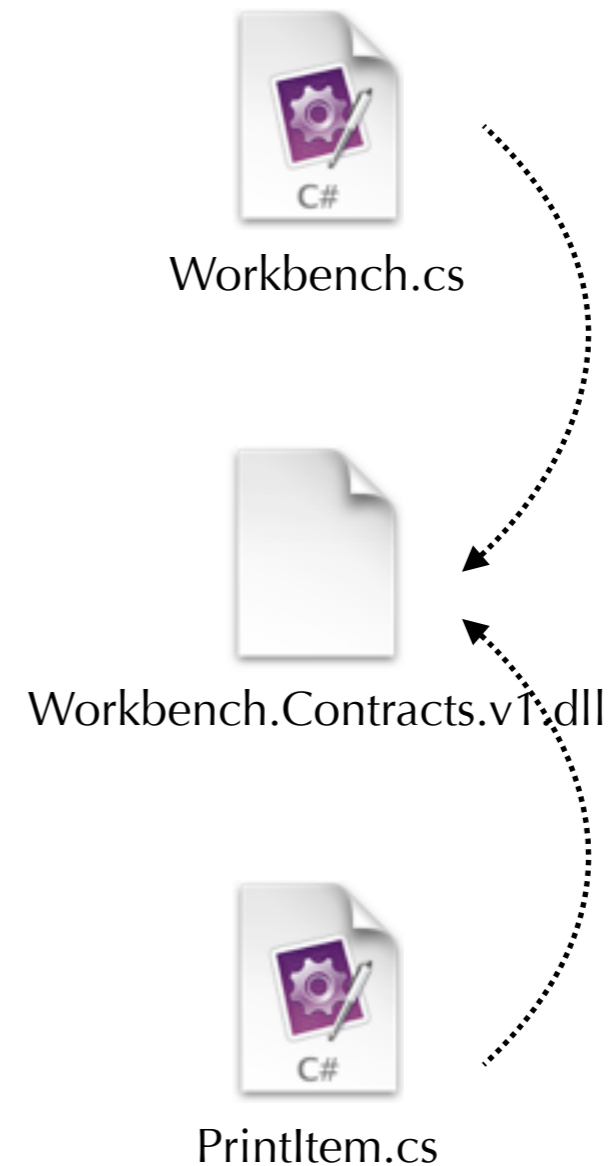


Versionierung (1)

```
[Extension(Name="Workbench", Slot="Startup")]  
[Isolation(AppDomain)]  
[Domain("Workbench")]  
class Workbench : IStartup {  
    ...  
}
```

```
[assembly: AssemblyVersion("1.0.*")]  
[Slot("MenuItem")]  
[Property(typeof(MenuItemAttribute))]  
interface IMenuItem {  
    ...  
}
```

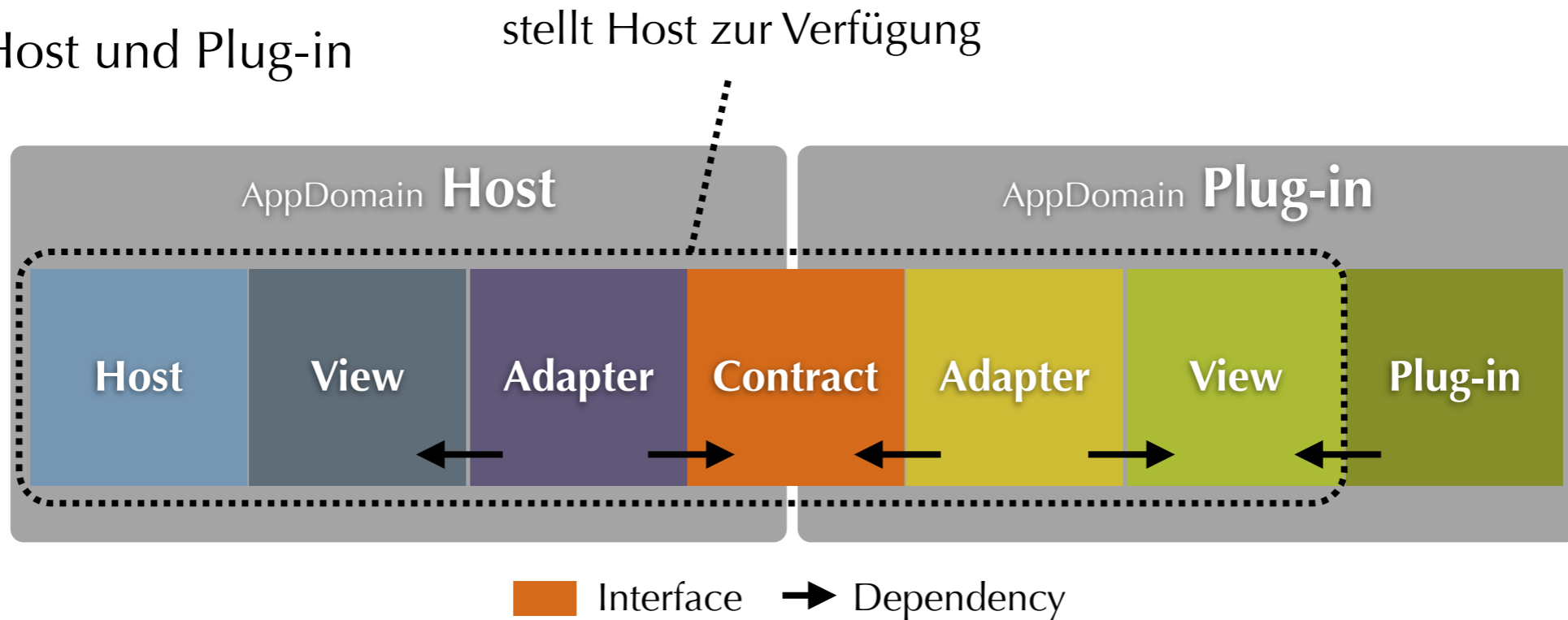
```
[Extension(Name="PrintItem", Slot="MenuItem")]  
[MenuItem(Text="Print", Icon="Print.ico")]  
[Domain("Print")]  
class PrintItem : IMenuItem {  
    ...  
}
```



Versionierung (2)

Stable Contract

zwischen Host und Plug-in



Abwärtskompatibilität

- Host v3 kann Plug-ins laden die als Ziel Host v1, v2 oder v3 hatten

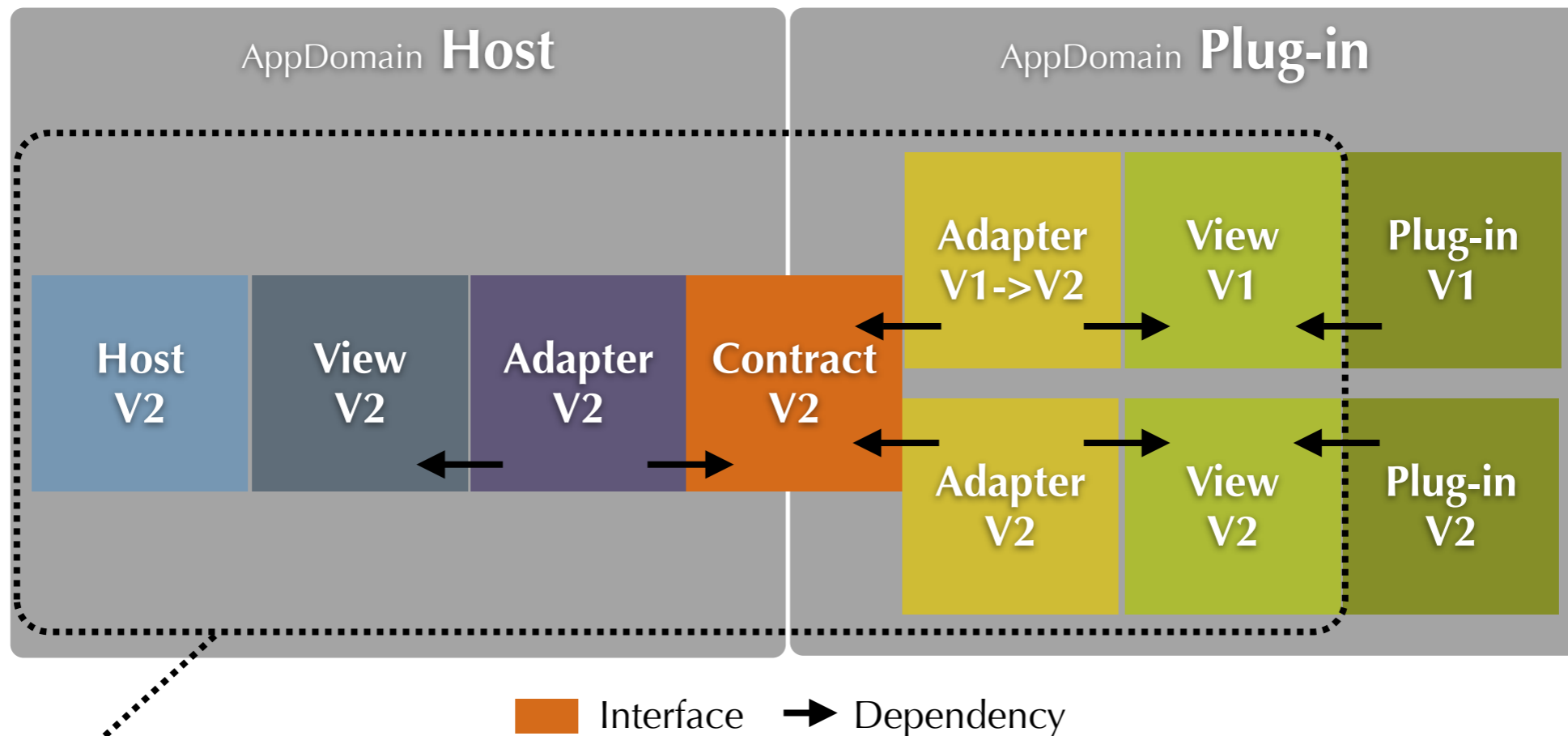
Aufwärtskompatibilität

- Host v1 kann Plug-ins laden die als Ziel Host v1, v2 oder v3 hatten

Versionierung (3)

Host / Plug-in Kompatibilität

Neuer Host mit Abwärtskompatibilität, altes Plug-in



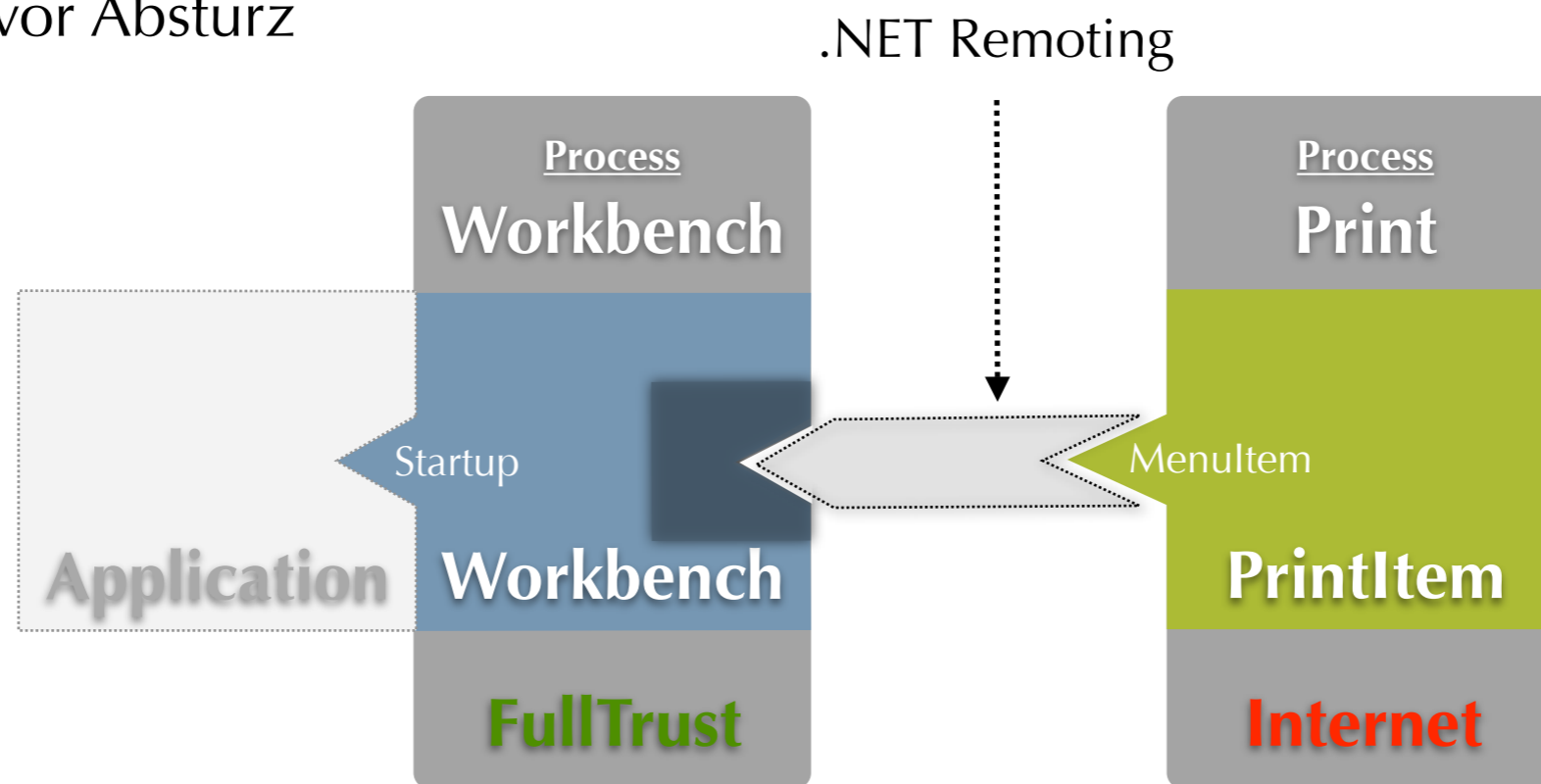
stellt Host zur Verfügung

Integrationsmodelle (4)

Process Isolation

Plug-in in separatem Prozess aktivieren

- schützt Host vor Absturz



```
[Extension(Name="Workbench", Slot="Startup")]  
[Isolation(Process)]  
[Domain("Workbench.Workbench")]  
class Workbench : IStartup {  
    ...  
}
```

```
[Extension(Name="PrintItem", Slot="MenuItem")]  
[Domain("Print.Print")]  
class PrintItem : IMenuItem {  
    ...  
}
```

Zusammenfassung

Eine Plug-in Architektur für Unternehmenssoftware soll Offenheit ermöglichen aber zugleich zuverlässig und sicher sein.

- Erweiterbarkeit und individuelle Arbeitsumgebung
- Sicherheit, Zuverlässigkeit, Versionierung
- nutzt .NET Features (Attribute, Metadaten, AppDomains, Security)
- vier Modelle zur Integration von Host und Plug-in

Danke für die Aufmerksamkeit.