Adding genericitiy to a plug-in framework

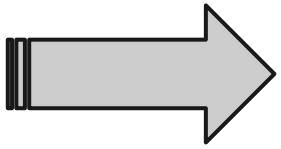
Wissenschaftliches Arbeiten - SS13 - Florian Oeser

agenda

- Plugin's in der Softwareentwicklung
- Plux.NET
- Generische Plug-in's

SE ohne Plugin's

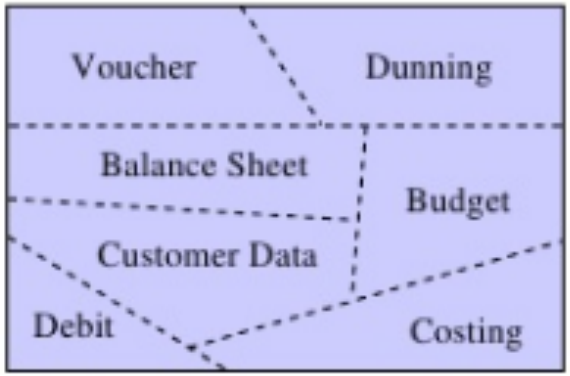
- Programme meist monolithisch
  - teuer
  - groß und komplex
  - schlecht wartbar
  - kleine Änderungen ⇨ Neuauslieferung
  - schwer erweiterbar

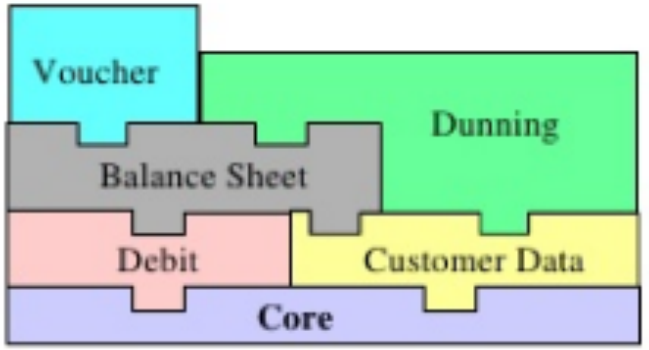


Flexibilität durch Plugin's!

Definition: Plugin

"Im Bereich der SE spricht man von einem Plugin, wenn eine oder mehrere Softwarekomponenten eine bestehende Anwendung um eine bestimmte Funktionalität erweitern."





Vom Monolithen...

... zu einem schlanken Kern + Plugins.

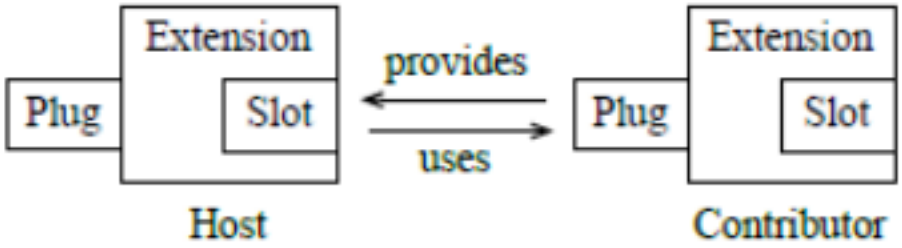
SE mit Plugin's

Plugin-Framework

- jeder Benutzer läd das was er braucht
- dynamisches Hinzfügen/Entfernen ohne Neustart
- einfache Erweiterbarkeit (Plug'n'Play) ohne programmieren/konfigurieren
- sichere Erweiterbarkeit
  - Was darf ein Plugin?
  - Wer darf ein Plugin hinzufügen?

Plux.NET

- Leichtgewichtiges Plugin-Framework für .NET
- Deklarative Komposition
  - über Metadaten (.NET Attribute) im Source
- Komponenten sind in sich geschlossen
  - keine separaten Konfigurationsdateien
- Deployment & Discovery
- ...



Host ⇨ öffnet Slot und stellt Funktionalitäten bereit

Contributor ⇨ stellt Plug bereit und bezieht Funktionalitäten

Slot-Plug-Beziehung zweier Extensions

```
[SlotDefinition("Logger")]
[ParamDefinition("TimeFormat", typeof(string))]
public interface ILogger {
 void Print(string msg);
}
```



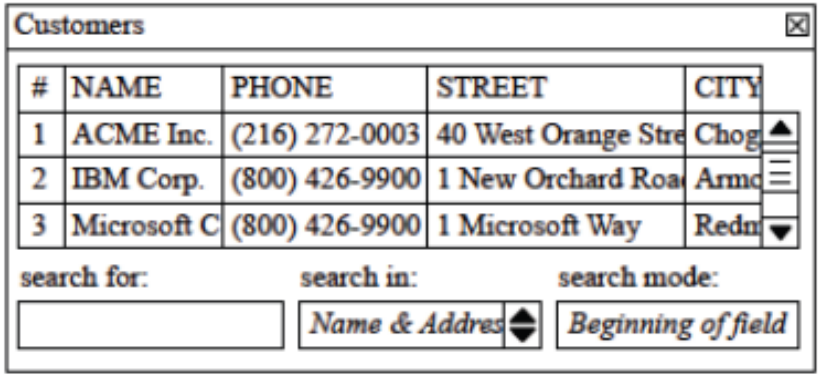
```
[Extension("ConsoleLogger")]
[Plug("Logger")]
[Param("TimeFormat", "hh:mm:ss")]
public class ConsoleLogger : ILogger {
 public void Print(string msg) {
  Console.WriteLine(msg);
 }
}
```

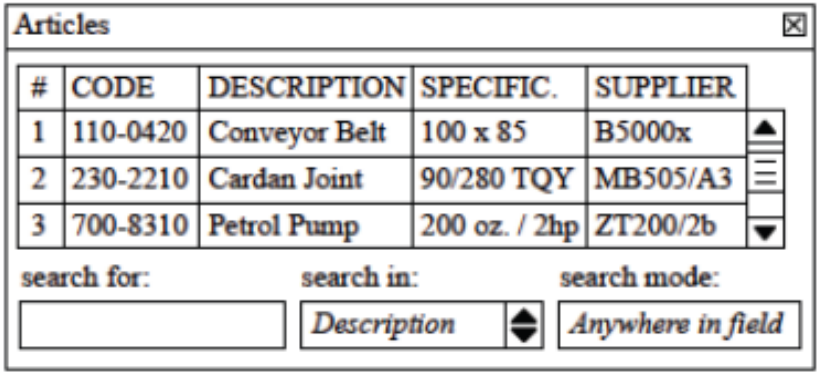
Slot-Plug Source Sample

Plux.NET-Anwendung

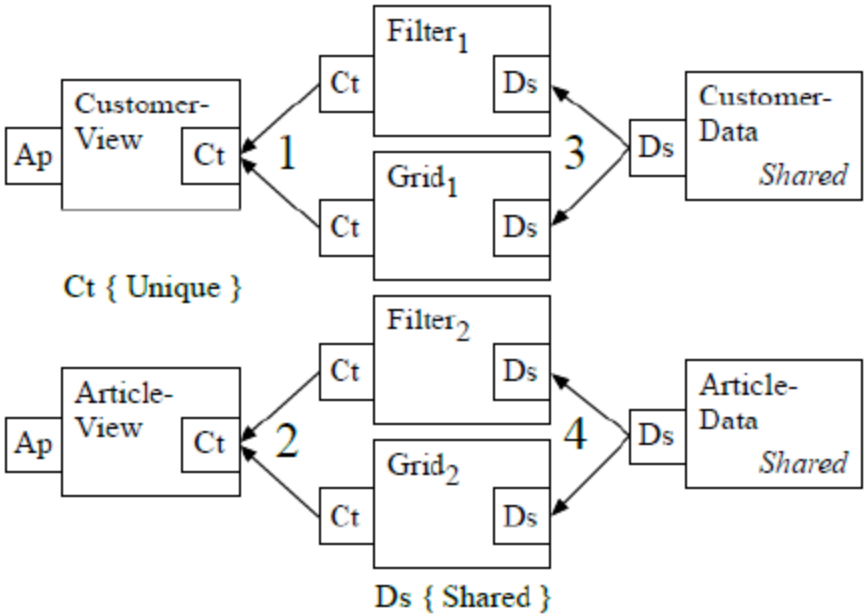
Deployment & Discovery

- Slots, Plugs und Anwendung
  - .NET Assemblies (*.dll)
- Deployment in spezielle Verzeichnisstruktur
- Automatische Komposition durch Framework
  - deklarierte Metadaten + Reflexion aus Assembly
- Plugin-basiert und flexibel

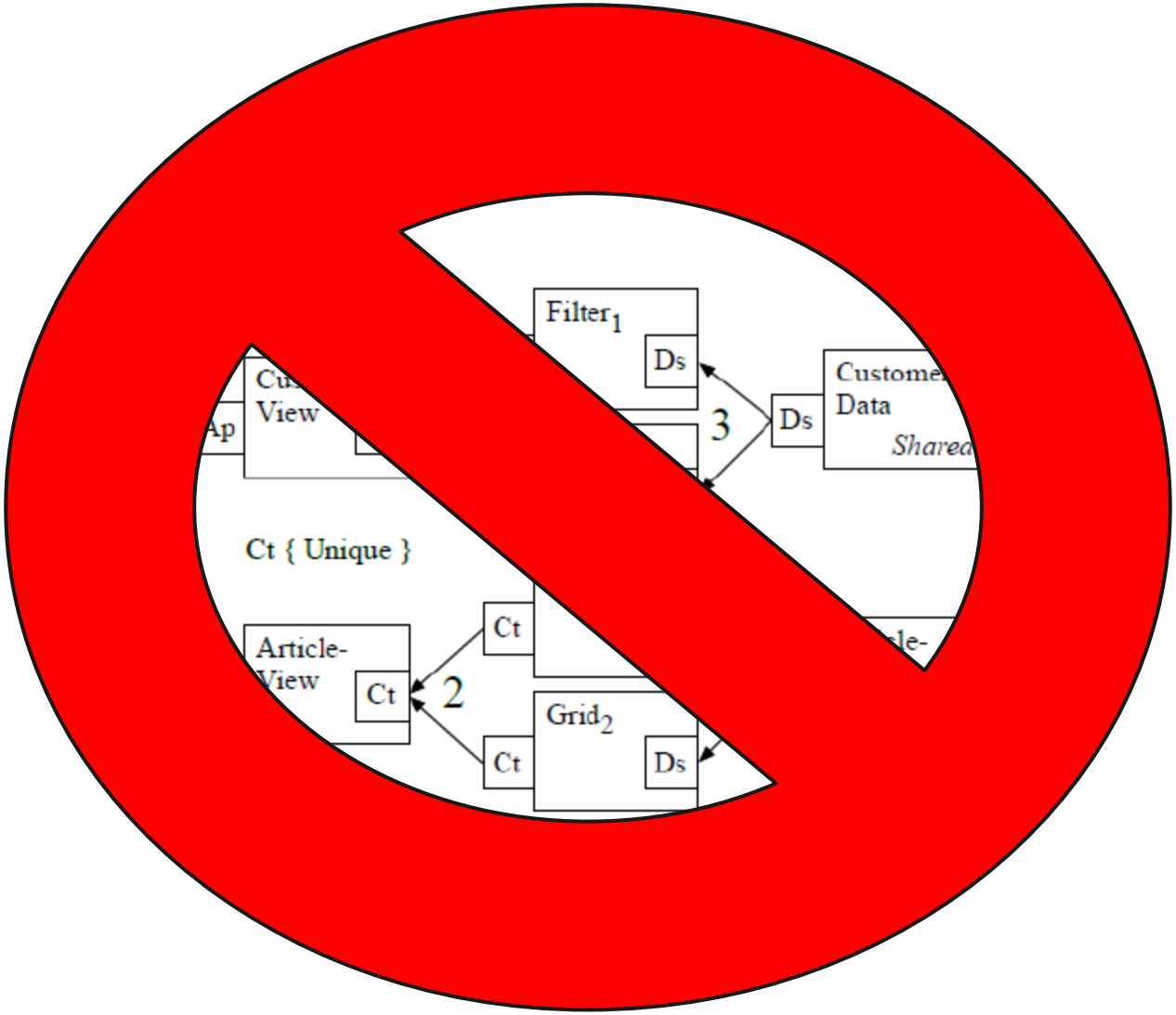




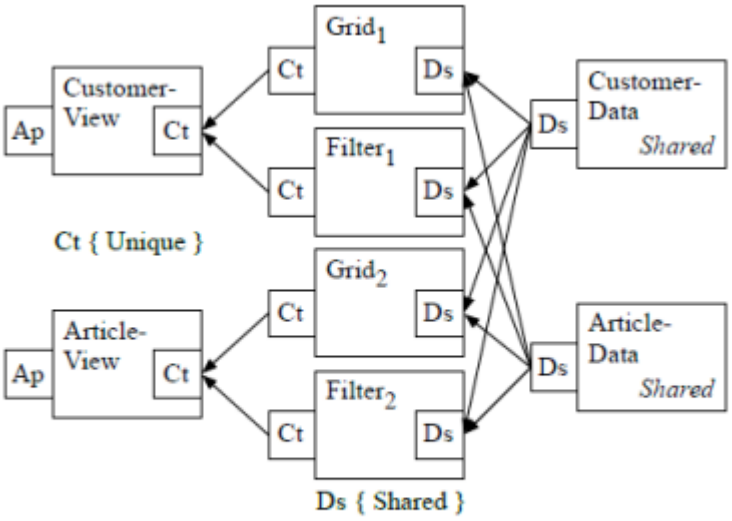
Generische Plugins?



Generische Plugins?



Generische Plugins?




```
[SlotDefinition("Control")]
[Param("Order", typeof(float))]
public interface IControl {
  Control Control { get; }
  string Name { get; }
}

[SlotDefinition("DataSource")]
public interface IDataSource {
  string Name { get; }
  object Data { get; }
  event EventHandler Changed;
}

[Extension]
[Plug("Control")]
[Param("Order", 0.5f)]
[Slot("DataSource", Shared=true)]
public class Grid : IControl { ... }
```

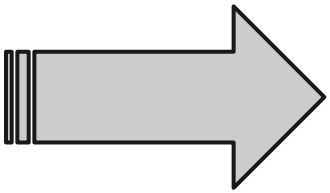
Generische Plugins?

Generische Plugins?

1. Automatische Komposition deaktivieren
   - widerspricht Plugin-Idee
2. Verschiedene Metadaten + Subklassen
   - automatische Komposition möglich
   - viele unnötige Typen

```
[Plug("ArticleControl"), Param("Order", 0.5f),
      Slot("ArticleData", Shared=true)]
public class ArticleGrid : Grid { ... }

[Plug("CustomerControl"), Param("Order", 0.7f),
      Slot("CustomerData", Shared=true)]
public class CustomerGrid : Grid { ... }
```

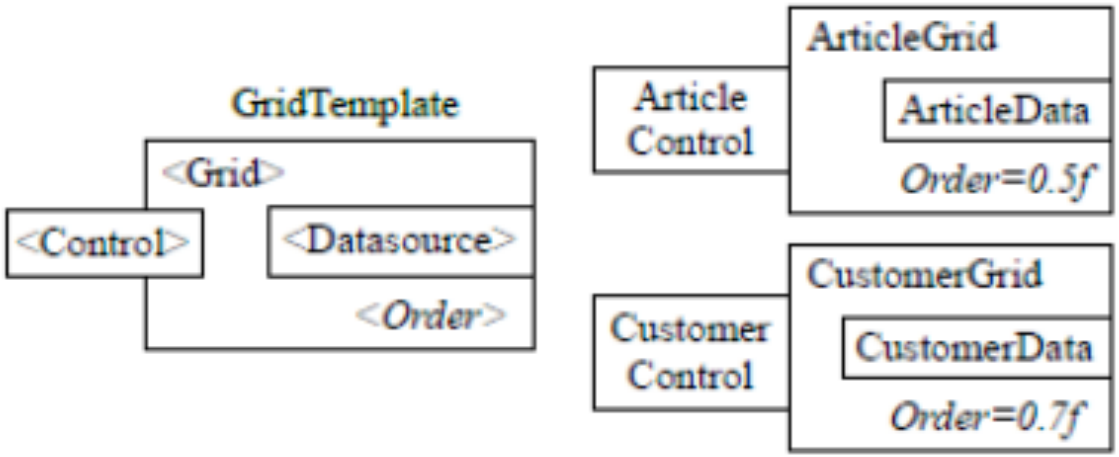


Generische Plugins!

Subclassing? Nein, danke.

Generische Plugins in Plux.NET

- Extensions mit Metadaten generieren
- Extension Templates enthalten
  - die eigentliche Klasse
  - Platzhalter für bekannte Metadaten
- Extension Templates sind unvollständig
  - Platzhalter müssen ersetzt werden
  - Platzhalternamen in spitzen Klammern
  - verschiedene Quellen denkbar



a) Template extension

c) Generated extensions

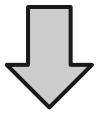
```
Grid       → ArticleGrid
             = Grid.dll/Grid
Control    → ArticleControl
Datasource → ArticleData
Order      → 0.5f
```

```
Grid       → CustomerGrid
             = Grid.dll/Grid
Control    → ArticleControl
Datasource → ArticleData
Order      → 0.7f
```

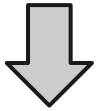
b) External metadata

Generische Extensions (schematisch)

```
[Template("Grid")]
[Plug("<Control>")]
[Param("Order", "<Order>")]
[Slot("<DataSource>", Shared=true)]
public class Grid : IControl { ... }
```



```
<Grid> -> ArticleGrid=Grid.dll/Grid
<Control> -> ArticleControl
<Datasource> -> ArticleData
<Order> -> 0.5f
```



```
[Extension("ArticleGrid")]
[Plug("ArticleControl")]
[Param("Order", 0.5f)]
[Slot("ArticleData", SlotDefinition="DataSource",
      Shared=true)]
public class Grid : IControl { ... }
```

Generische Extensions (Source)

Deployment & Discovery

- Template Analyzer
  - sucht *[Template]*-Attribut
  - sucht Konfigurationsdatei mit Metadaten
  - ebenfalls Plugin-basiert
- Austausch von Metadaten-Platzhaltern
  - Slot-, Plug- und Parameter-Attribute

Zusammenfassung

- Wiederverwendung von Komponenten
  - Grid's mit unterschiedlichen Views und Daten
- Plugins müssen flexibel zusammengesetzt werden
  - ohne hohen Aufwand und Nachteile
- Template-basierter Ansatz
  - Konkretisierung erst zur Laufzeit

Quellen

- Paper: Adding genericity to a plugin framework
- Präsentation: Plux.NET (Software-Komposition durch Plug & Play)
  - http://de.slideshare.net/foerderverein/folien-mssenbck

Ende. Danke.

Fragen?