14.–17. 09. 2009
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Flexibel sein

Konfigurierbare Strukturen und Logik in .NET-Anwendungen

Rainer Stropek

cubido business solutions gmbh

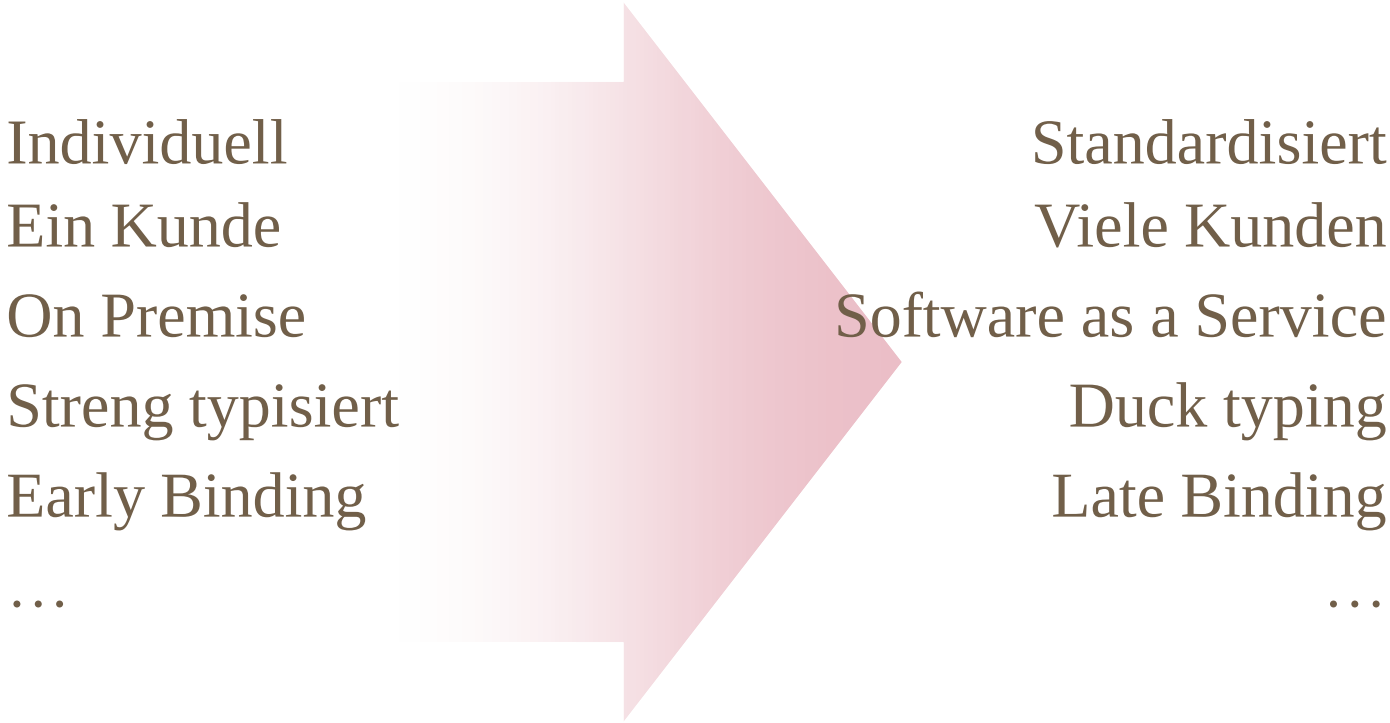
Inhaltsbeschreibung

Der Schlüssel zur Umsetzung von konfigurierbaren Applikationen mit variablem Datenmodell sind Metadaten. Im .NET-Framework gibt es seit den ersten Versionen Ansätze dazu (z. B. `ICustomTypeDescriptor`). In .NET 2.0 wurde dieser Mechanismus drastisch erweitert (`TypeDescriptor` und `TypeDescriptionProvider`). Durch die Dynamic Language Runtime (DLR) findet nun ein neues Konzept für dynamische Bindung und Metadaten seinen Weg in das .NET-Framework (`IDynamicObject`).

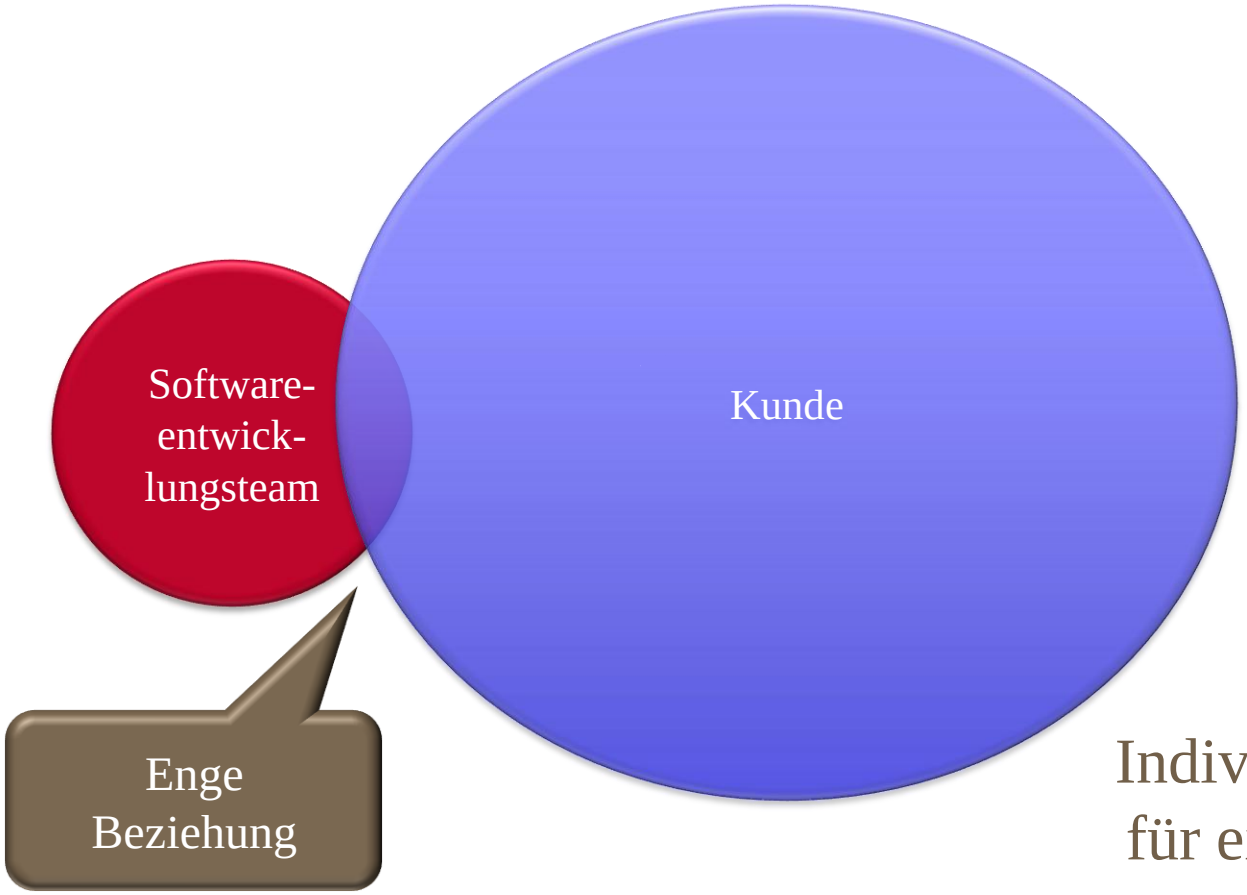
In diesem Vortrag wird anhand eines durchgängigen Beispiels gezeigt, wie bereits mit den heute verfügbaren Mitteln (d. h. mit C# 3.x, .NET Framework 3.5, DLR, ANTLR und XAML/WPF für das UI) eine Architektur für hochgradig anpassbare .NET-Applikationen aussehen kann. Dabei liegt der Schwerpunkt jedoch nicht auf der Vermittlung von Implementierungsdetails sondern darauf, zu erklären und praktisch zu zeigen, wie eine Architektur für anpassbare Datenstrukturen und Logik in .NET aussehen kann. Dieser Vortrag richtet sich damit an alle Entwickler, die konfigurierbare Anwendungen für ein breites Zielpublikum erstellen müssen. Er zeigt wie man kundenspezifische Anpassungen der Datenstrukturen ermöglicht und wie man mit wenig Aufwand zu einer Script-Sprache für die eigene Applikation kommt. Es wird gezeigt, dass dies schon jetzt – noch ohne C# 4.0 – praktisch einsetzbar ist und so geschrieben werden kann, dass der Code nahezu ohne Änderung bei Verfügbarkeit von VS 2010 mit dem dynamic-Datentyp von C# 4.0 konsumiert werden kann.

Worum geht es?

Worum geht es?

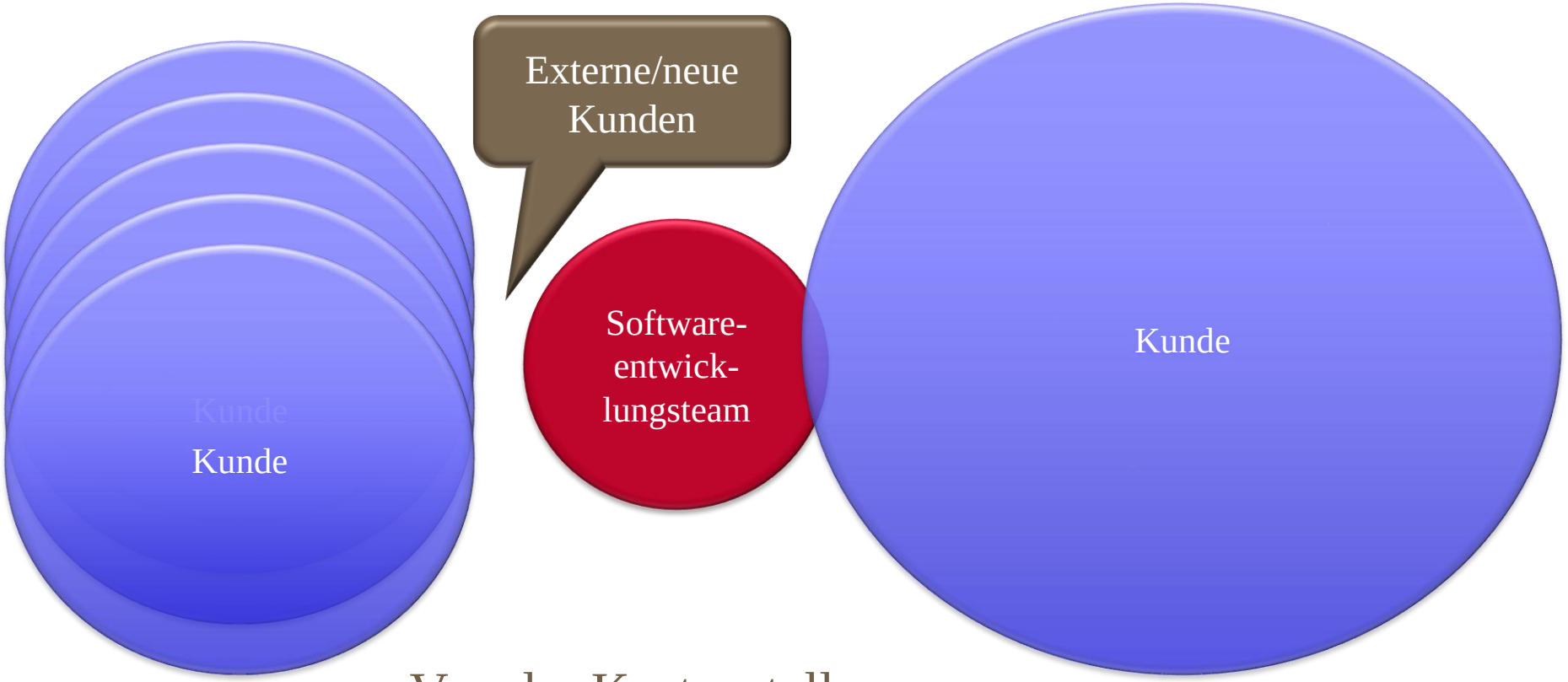


Ausgangspunkt



Individuelle Entwicklung
für einen großen Kunden
(intern oder extern)

Ausgangspunkt

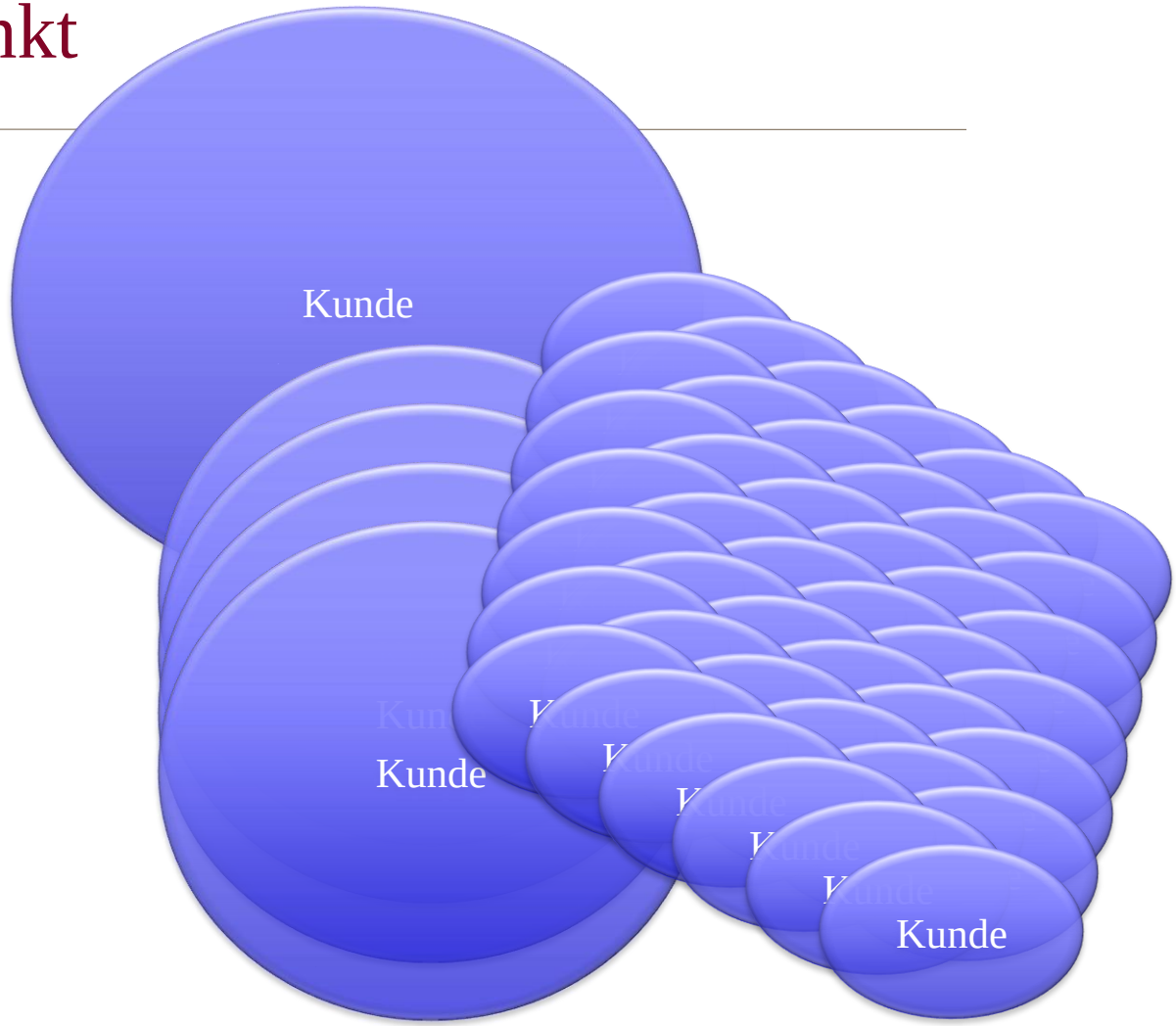


Von der Kostenstelle zum Profitcenter…

Ausgangspunkt

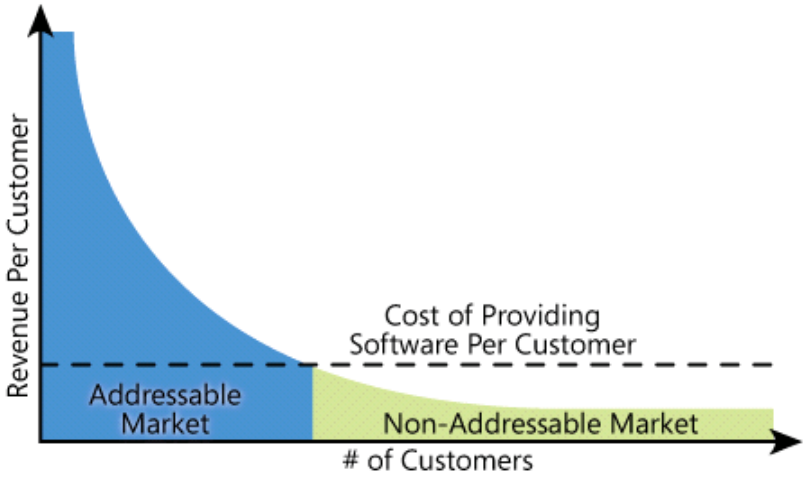
Standardisiertes Massenprodukt

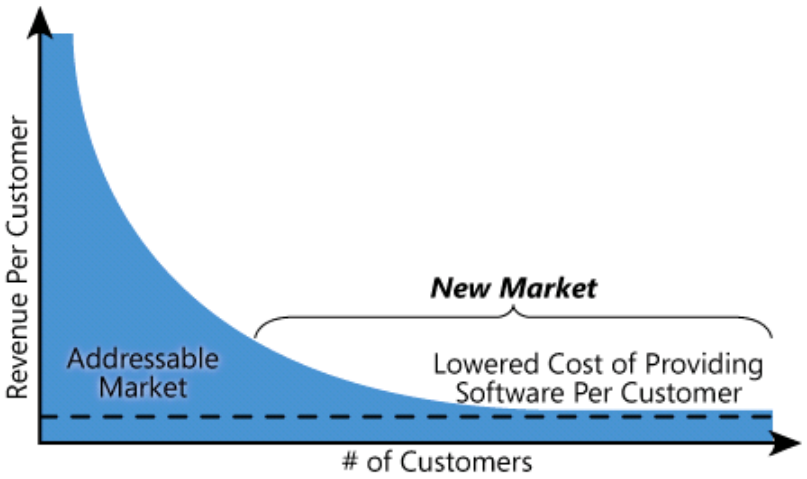
Software-entwick-lungsteam



Den dicken Hund am Schwanz packen…

Ausgangspunkt





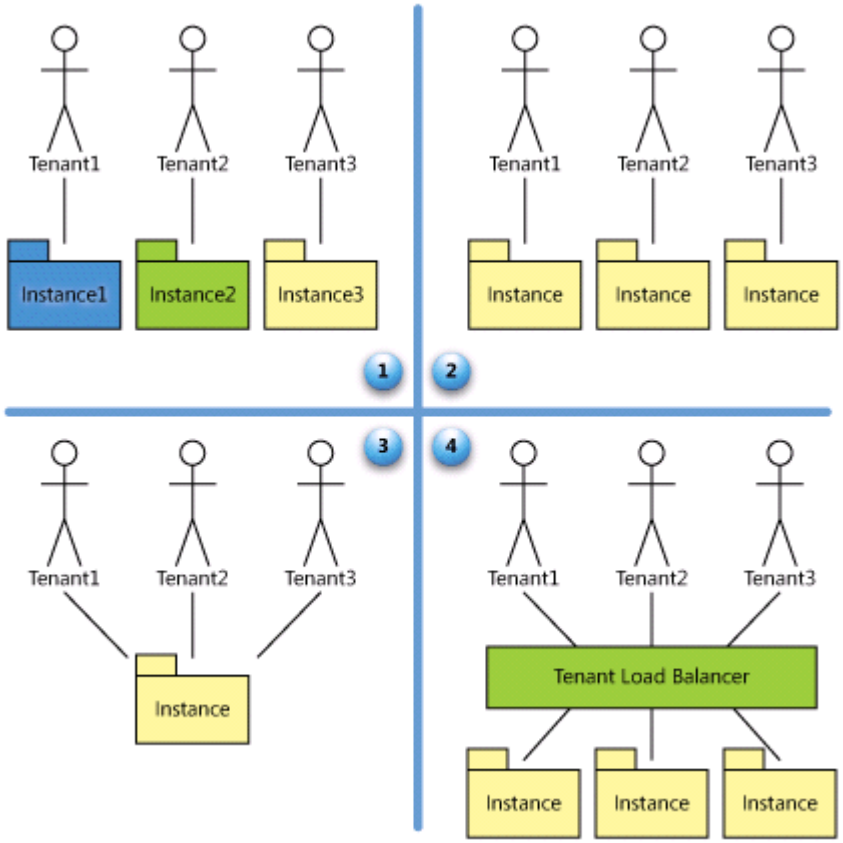
The long tail

http://www.wired.com/wired/archive/12.10/tail.html

Die Probleme

One Size Does Not Fit All!

Die Probleme – Multi-Tenancy

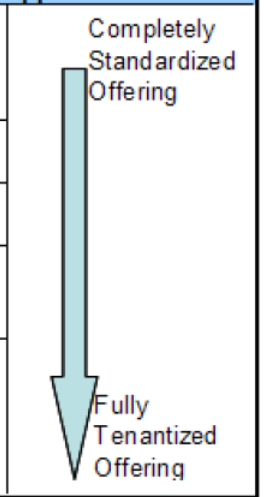


SaaS Maturity Levels

Kwok, Nguyen, Lam: A Software as a Service with Multi-tenancy Support for an Electronic Contract Management Application, IEEE International Conference on e-Business Engineering, pp. 179-186, 2008

Die Probleme – Erweiterbarkeit und Anpassbarkeit

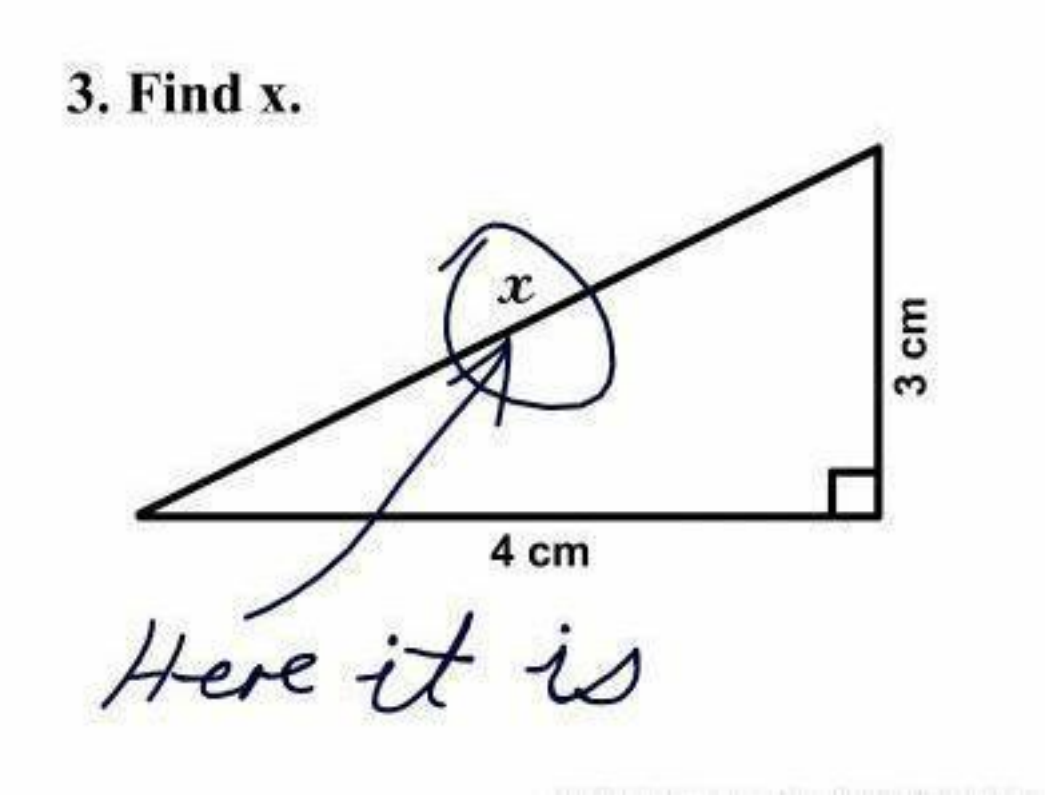
Level of Competency	Description	Approach	Variance Level Supported
Entry	Highly standardized offering without any configuration and customization support	Well design the functionalities as standardized offering to cover targeted customers	None
Aware	Relatively standardized offering with pre-defined variance points	Offer parameterized configuration	Low
Capable	Relatively standardized offering with user defined configuration	Offer self serve configuration tool to empower customers	Medium
Mature	Base offering with programable enviroment to enable user preferred customization	Offer scripting based programming for very flexible customization	High
World Class	Offer a platform supported by programming model and tools to enable extremely strong customization or even new application development	Offer well defined programming model and tools to enable extensive customization and new application development	Extremely High



SaaS Configuration Competency Model

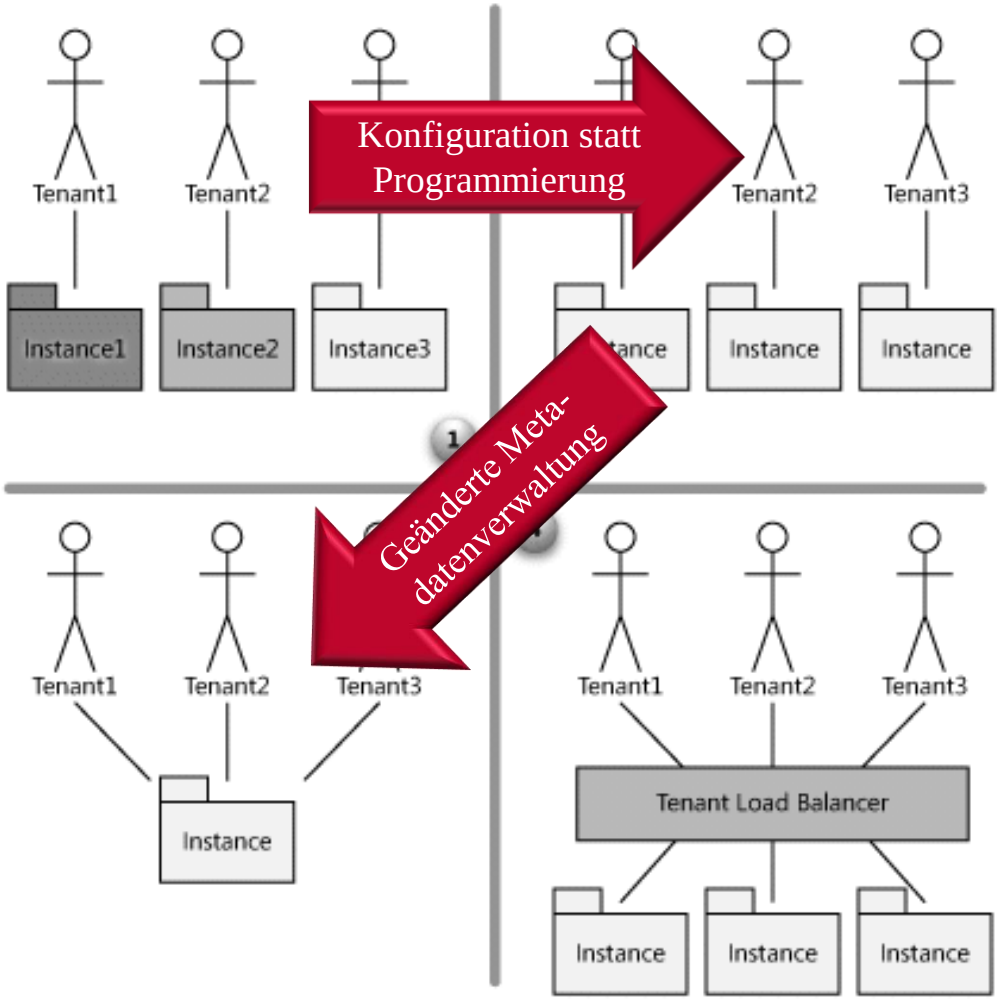
Wei Sun, Xin Zhang, Chang Jie Guo, Pei Sun, Hui Su: Software as a Service: Configuration and Customization Perspectives, in Proceedings of IEEE Congress on Services Part II, 2008

Lösungsansätze

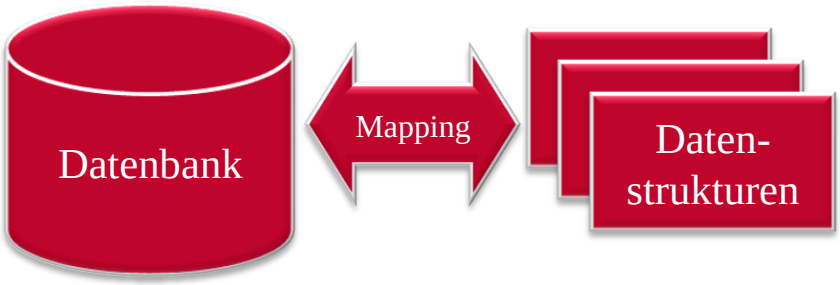


Quelle: http://www.flickr.com/photos/dullhunk/426622486/

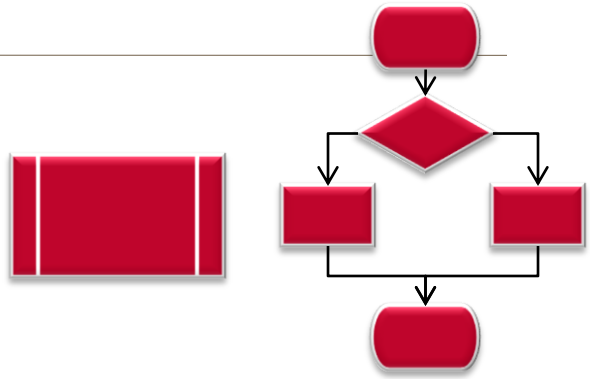
Metadata Rulez!



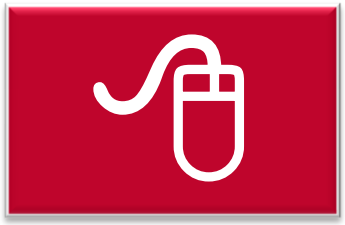
Metadata Rulez!
Anwendungsbereiche



Datenmodell



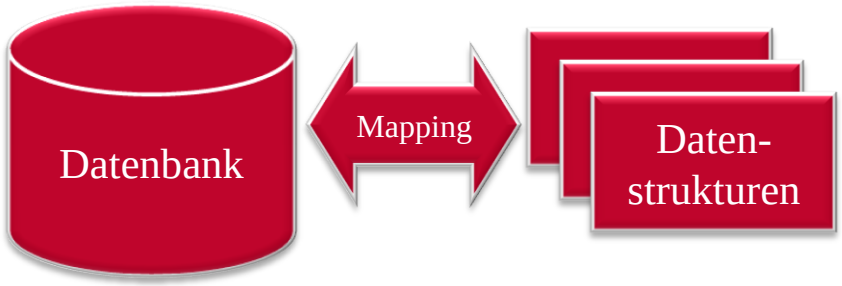
Geschäftsprozesse
und -logik



Benutzerschnittstelle

Authentifizierung und
Authorisierung

Datenmodell

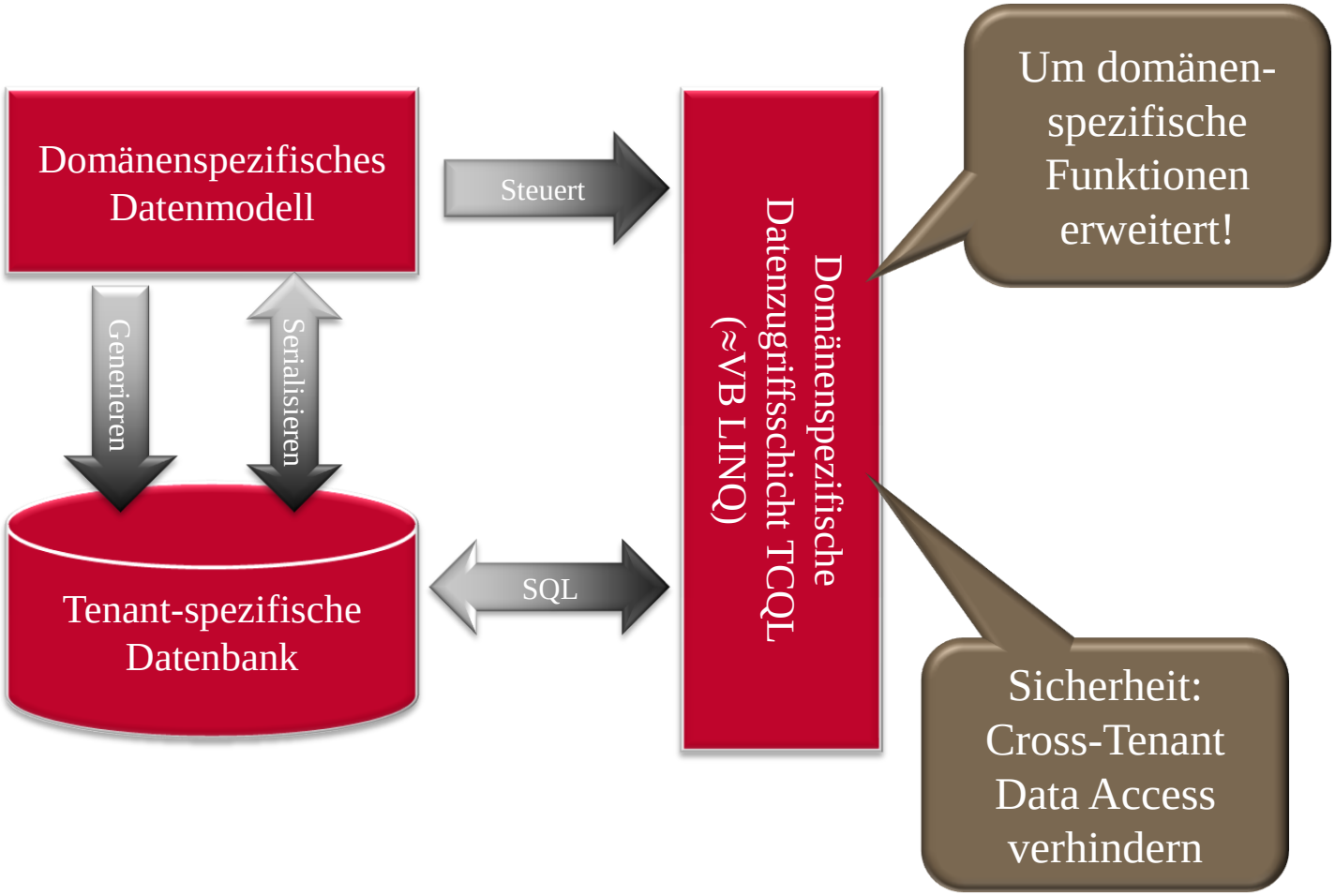


Wozu diskutieren? ORMs gibt es genug!

- Vorteil von ORMs = Nachteil bei SaaS
  - Strenge Typsicherheit durch generierten Code
  - Early Binding
  - Beispiel: Microsoft Entity Framework + LINQ

SaaS bedeutet oft Programmieren gegen ein (teilweise) unbekanntes Datenmodell!

Unser Ansatz im Bereich Datenmodell



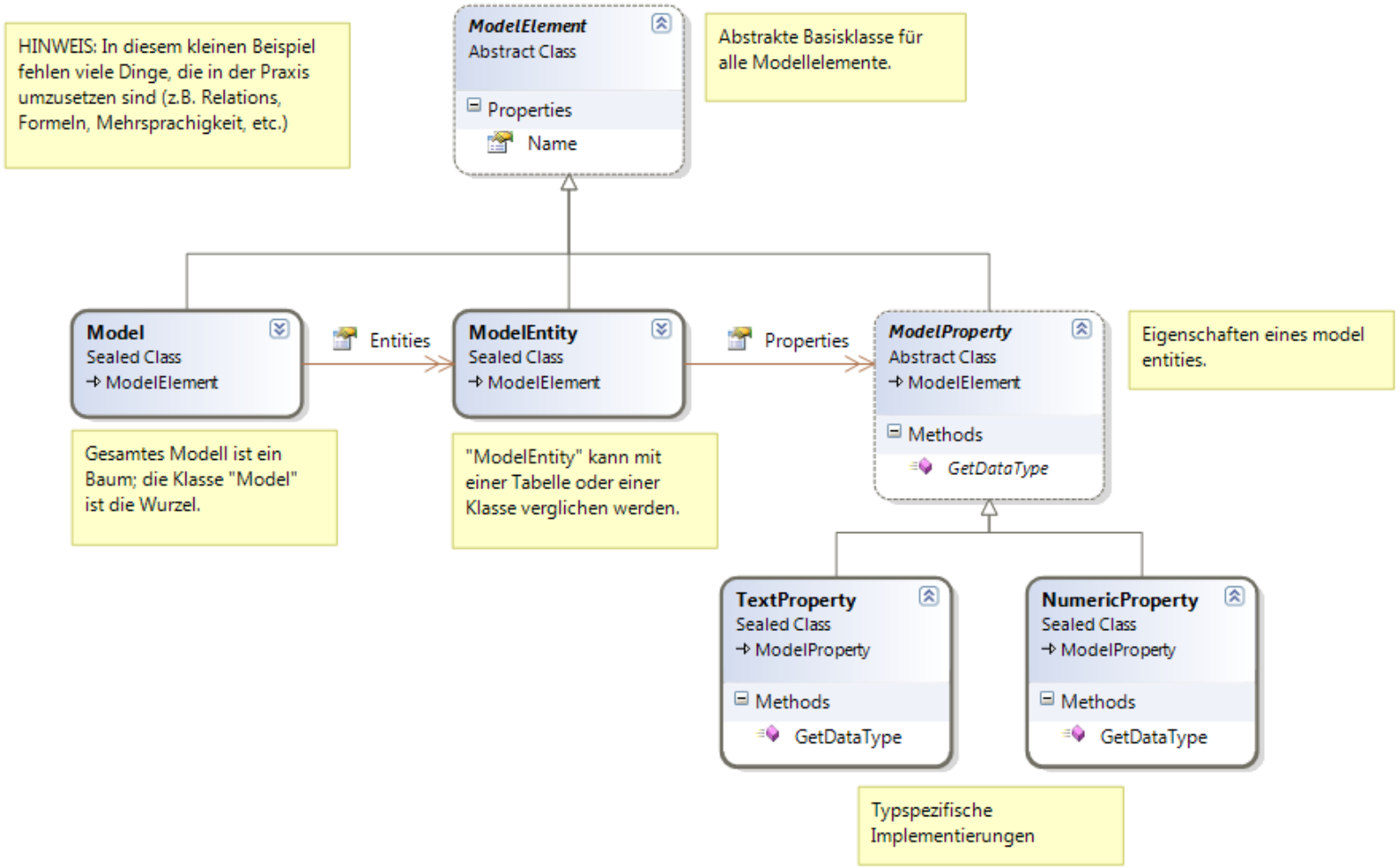
Demo 1

Domänenspezifisches Metadatenmodell

Code Snippet
XAML Metadatenmodell

```
<Model xmlns="clr-namespace:DlrSample.Data.DataModel;assembly=DlrSample.Data">
     <Model.Entities>
          <ModelEntity Name="Customer">
               <ModelEntity.Properties>
                    <TextProperty Name="CustomerName" />
                    <TextProperty Name="CustomerDescription" />
               </ModelEntity.Properties>
          </ModelEntity>
          <ModelEntity Name="Project">
               <ModelEntity.Properties>
                    <TextProperty Name="ProjectCode" />
                    <TextProperty Name="ProjectManager" />
                    <NumericProperty Name="Budget" />
                    <NumericProperty Name="CumulatedCosts" />
               </ModelEntity.Properties>
          </ModelEntity>
     </Model.Entities>
</Model>
```

UML Klassendiagramm
XAML Metadatenmodell



Code Snippet
Generieren von SQL mit StringTemplate

```
group DataModelTemplates;

CreateTable(context, entity) ::=
<<
CREATE TABLE [<context.Tenant>].[<entity.Name>]
(
     <entity.Name>Uuid uniqueidentifier NOT NULL,
     <entity.Properties:{ p | <p:(p.Type.Name)()>}; separator=",\n">
)
ALTER TABLE [<context.Tenant>].[<entity.Name>] ADD CONSTRAINT
     DF_<entity.Name>_<entity.Name>Uuid DEFAULT newid() FOR <entity.Name>Uuid

ALTER TABLE [<context.Tenant>].[<entity.Name>] ADD CONSTRAINT PK_<entity.Name>
PRIMARY KEY CLUSTERED (  <entity.Name>Uuid )
>>

TextProperty(property) ::= "<property.Name> varchar(50) NULL„
NumericProperty(property) ::= "<property.Name> numeric(18,4) NULL"
```

Code Snippet
Generieren von SQL mit StringTemplate

```
private void CreateModelEntity(StringTemplateGroup stg,
    DbCommand cmd, ModelEntity entity)
{
     var template = stg.GetInstanceOf("CreateTable");
     template.SetAttribute("context", this);
     template.SetAttribute("entity", entity);
     cmd.CommandText = template.ToString();
     cmd.ExecuteNonQuery();
}
```

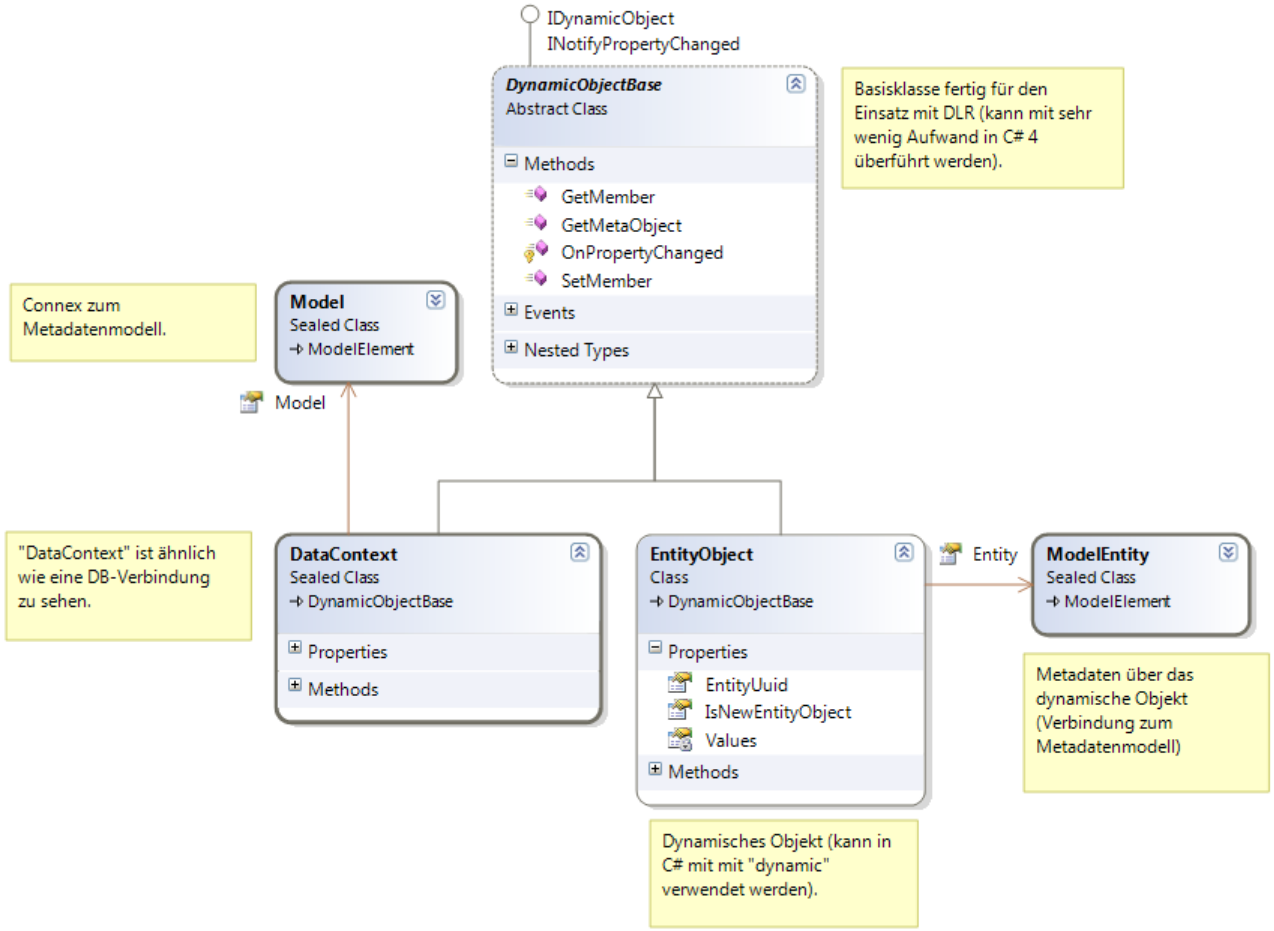
Tipps zum domänenspezifischen Datenmodell

- XAML eignet sich gut
  - Vorteil gegenüber XML: Kein extra Schema notwendig
  - Vorteil gegenüber Scripting: Code Injections leichter zu verhindern
- SQL-Generierung mit Template Engine
  - Hier gezeigt: StringTemplate von ANTLR
- Datenbankunabhängigkeit leicht erreichbar
- Konventionen über Templates umsetzen
- Auf spezifische Anforderungen der jeweiligen Domäne eingehen
  - Z.B. Funktionen, Mehrsprachigkeit, etc.
- Multi-Tenancy beachten

Demo 2

Dynamisches Datenmodell

UML Klassendiagramm Datenverwaltung



Code Snippet
Generieren von SQL mit StringTemplate

```
group DataTemplates;

SaveObject(context, object) ::=
<<
<if(object.IsNewEntityObject)>
     insert into [<context.Tenant>].[<object.Entity.Name>] (
          <object.Entity.Name>Uuid,
          <object.Entity.Properties:{ p | <p.Name>}; separator=", ">
     ) values (
          @Uuid,
          <object.Entity.Properties:{ p | @<p.Name>}; separator=", ">
     )
<else>
     update [<context.Tenant>].[<object.Entity.Name>]
     set <object.Entity.Properties:{ p | <p.Name> = @<p.Name>}; separator=", ">
     where <object.Entity.Name>Uuid = @Uuid
<endif>
>>
```

Code Snippet
Generieren von SELECT mit StringTemplate

```
group DataTemplates;

SelectAll(context, entity) ::=
<<
select x.<entity.Name>Uuid as Uuid, <entity.Properties:{ p | x.<p.Name>};
separator=", ">
from [<context.Tenant>].[<entity.Name>] x
>>
```

```
public IEnumerable<EntityObject> Select(ModelEntity entity)
{
     var result = new List<EntityObject>();

     using (var templateReader=DataContext.GetTemplateReader("SelectTemplates"))
     {
          var stg = new StringTemplateGroup(templateReader);
          var template = stg.GetInstanceOf("SelectAll");
          template.SetAttribute("context", this);
          template.SetAttribute("entity", entity);
          return this.ExecuteSelect(entity, template.ToString());
     }
}
```

Code Snippet
Dynamische Objekte

```
public class EntityObject : DynamicObjectBase
{
     […]
     private Dictionary<string, object> Values { get; set; }
     […]

     public override object GetMember(string name)
     {
          if ( this.Values.ContainsKey(name)) { return this.Values[name]; }

          var prop = this.Entity.Properties.Where(p => p.Name == name);
          if (prop.Count() > 0) {  return null; }

          return base.GetMember(name);
     }

     […]
}
```

Code Snippet
Anwendung dynamischer Objekte

```
var projectEntity = context.Model.Entities.Where(
     e => e.Name == "Project").First();
var eo = context.CreateEntityObject(projectEntity);
eo.SetMember("ProjectCode", "Test1");
eo.SetMember("Budget", 20);
eo.SetMember("CumulatedCosts", 18);
context.SaveObject(eo);

var result = context.Select(projectEntity);
var firstResult = result.First();
firstResult.SetMember("ProjectCode", "Test2");
context.SaveObject(firstResult);
```

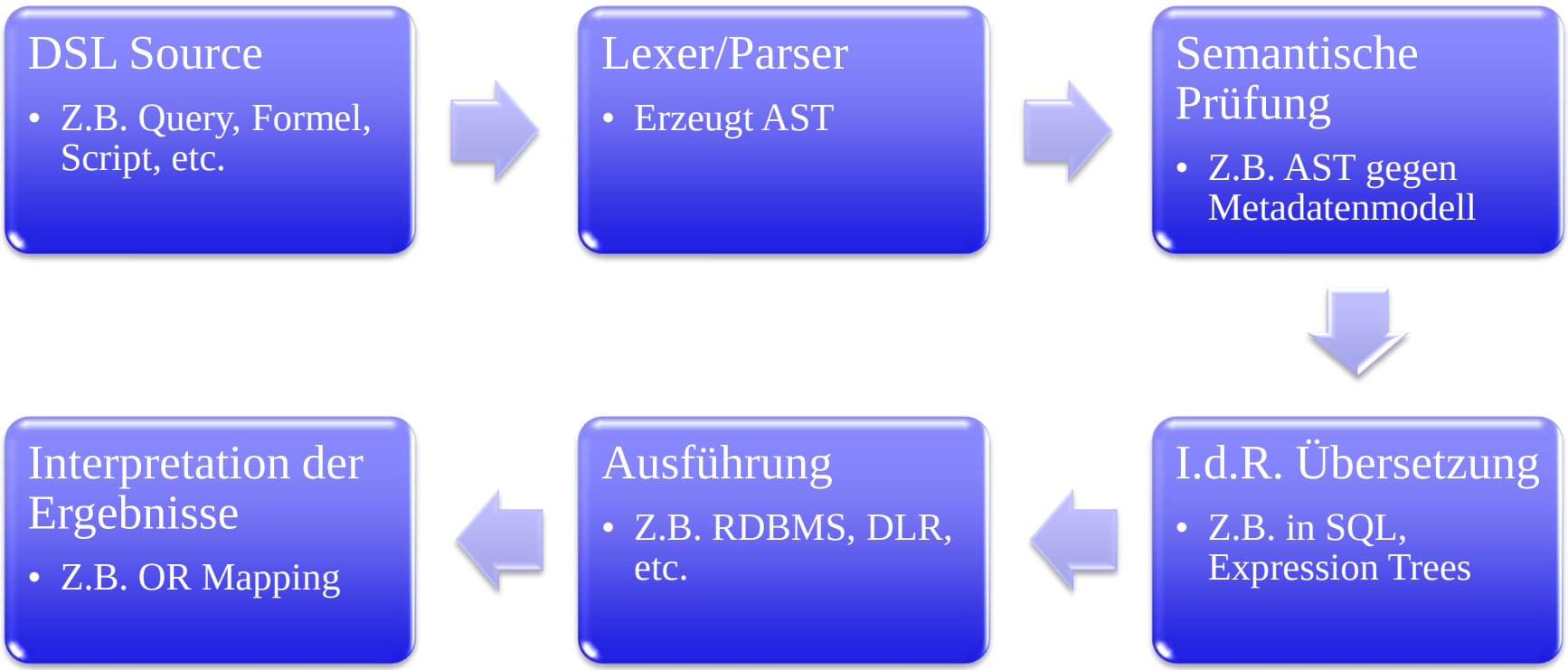
Tipps zum dynamischen Datenmodell

- Schon heute mit C# 3 + DLR umsetzbar!
  - Alles final, keine Betas
- Darauf achten, dass Umstieg auf C# 4 leicht machbar bleibt
  - Verwendung wird in C# 4 wesentlich eleganter!
  - Z.B. DLR-spezifische Dinge in eigenen Klassen/Moduln kapseln
- Durch Konventionen das Leben erleichtern
  - Z.B. jedes Objekt hat eine Uuid als Key
- Mischung aus dynamischem Datenmodell und strenger Typprüfung mit Interfaces machbar
- Dieses Beispiel war stark vereinfacht – Aufwand nicht unterschätzen!

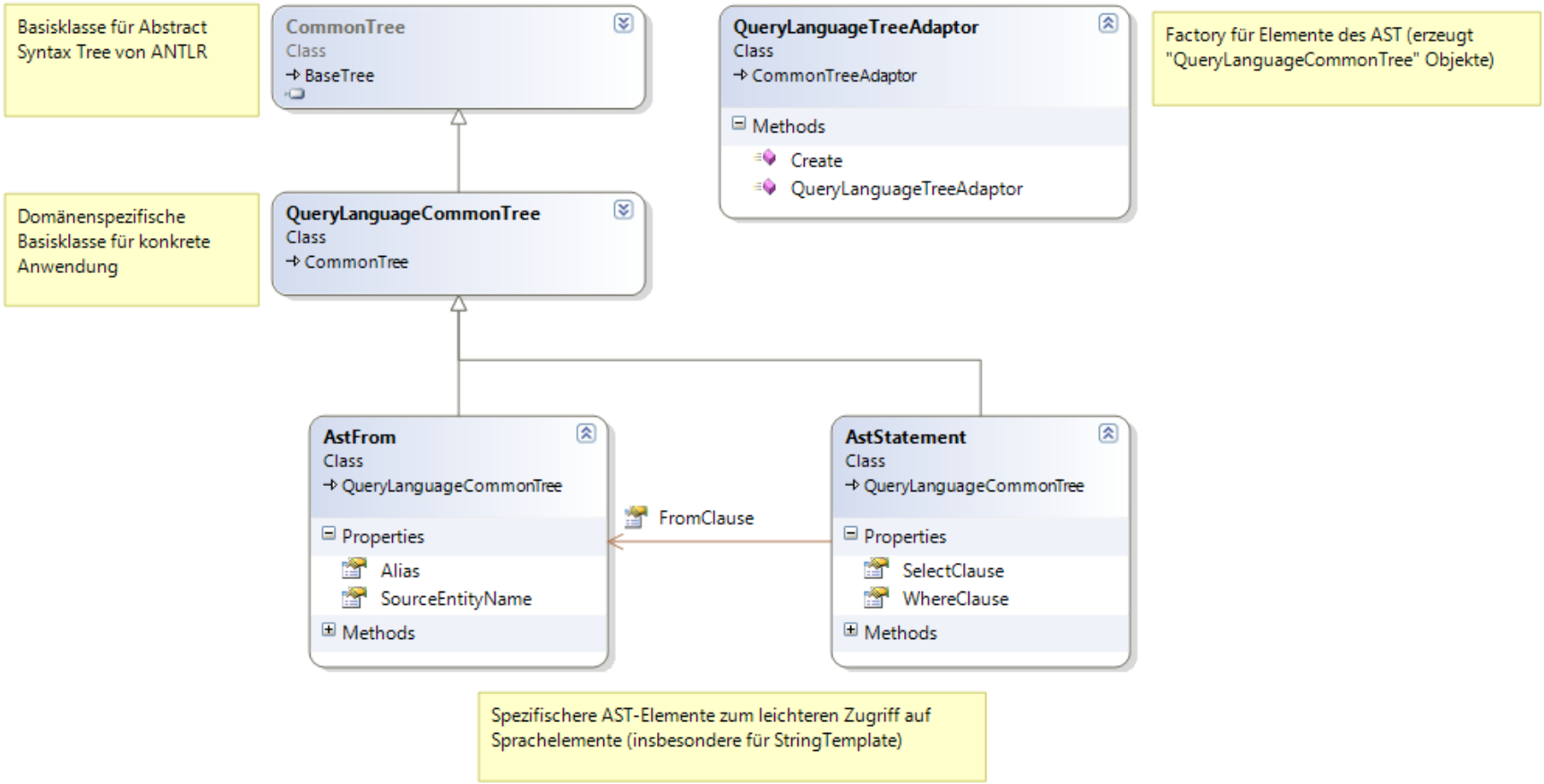
Demo 3

Domänenspezifische Abfragesprache

Prinzip von DSLs



UML Klassendiagramm Abfragesprache



Code Snippet
Tree Adaptor zum Aufbau des AST

```
internal class QueryLanguageTreeAdaptor : CommonTreeAdaptor
{
     […]
     public override object Create(IToken payload)
     {
          if (payload != null)
          {
               switch (payload.Type)
               {
                    case QueryLanguageLexer.STMT:
                         return new AstStatement(payload);
                    case QueryLanguageLexer.FROM:
                         return new AstFrom(payload);
                    default:
                         break;
               }
          }
               return new QueryLanguageCommonTree(payload);
     }
}
```

Code Snippet
Spezifische AST-Knoten

```
internal class AstFrom : QueryLanguageCommonTree
{
     […]
     public string Alias
     {
          get
          {
               return this.QueryLanguageChildren[0].Text;
          }
     }

     public string SourceEntityName
     {
          get
          {
               return this.QueryLanguageChildren[1].Text;
          }
     }
}
```

Code Snippet
Übersetzung in SELECT mit StringTemplate

```
Select(context, entity, ast) ::=
<<
select <ast.FromClause.Alias>.<entity.Name>Uuid as Uuid, <entity.Properties:{ p
| <ast.FromClause.Alias>.<p.Name>}; separator=", ">
from [<context.Tenant>].[<entity.Name>] <ast.FromClause.Alias>
<if(ast.WhereClause)>
where <first(ast.WhereClause.Children):callViaTypeName()>
<endif>
>>

callViaTypeName() ::= "<it:(it.TypeName)()>"

EQUAL() ::= "<first(it.Children):callViaTypeName()> =
<last(it.Children):callViaTypeName()>"
DIFFERENT() ::= "<first(it.Children):callViaTypeName()> <>
<last(it.Children):callViaTypeName()>"
AND() ::= "<first(it.Children):callViaTypeName()> and
<last(it.Children):callViaTypeName()>"
OR() ::= "<first(it.Children):callViaTypeName()> or
<last(it.Children):callViaTypeName()>"
MEMBERACCESS() ::= "<ast.FromClause.Alias>.<last(it.Children)>"
LITERAL() ::= "<it>"
```

Tipps zum domänenspezifischen Abfragesprache

- Keine eigenen Sprachen erfinden
  - Gut kopiert ist besser als schlecht erfunden
  - Security: Gut, um Möglichkeiten einzuschränken
  - Intellectual Property beachten!
- Domänenspezifische Dinge betonen
  - Den Anwendern das Leben erleichtern
- Abfrage- und Formelsprache in einem
  - Zwei Fliegen mit einer Klappe!
  - Kann für DB (SQL) übersetzt und in-memory ausgeführt (MS Expression Trees) werden
- Man braucht kein Doktorat in Compilerbau, um eine DSL umzusetzen
  - Basiskenntnisse schaden nicht → Literatur
  - Keine eigenen Compiler bauen!

Demo 4

Unterstützung für Scripts hinzufügen

Code Snippet
Scripting initialisieren – Konfiguration

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
     <configSections>
          <section name="microsoft.scripting„
               type="Microsoft.Scripting.Hosting.Configuration.Section,
Microsoft.Scripting, Version=0.9.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" />
     </configSections>

     <microsoft.scripting>
          <languages>
               <language names="IronPython;Python;py" extensions=".py„
                    displayName="IronPython v2.0„
                    type="IronPython.Runtime.PythonContext, IronPython,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
          </languages>
     </microsoft.scripting>
</configuration>
```

Code Snippet
Scripting initialisieren

```
private void InitializeScriptingEnvironment()
{
     this.ScriptRuntime = ScriptRuntime.CreateFromConfiguration();
               this.ScriptRuntime.LoadAssembly(Assembly.GetExecutingAssembly());

     this.ScriptScope = this.ScriptRuntime.CreateScope();
     this.ScriptScope.SetVariable("Context", this);
     var engine = this.ScriptRuntime.GetEngine("py");

     // Execute ImportScript
     var script = new StringBuilder();
     script.AppendLine("import clr");
     script.AppendLine(@"clr.AddReference(""DlrSample.Data"")");
     script.AppendLine("from DlrSample.Data.DataModel import Model, […]");

     engine.CreateScriptSourceFromString(script.ToString(),
          SourceCodeKind.Statements).Execute(this.ScriptScope);
}
```

Code Snippet
Scripts ausführen

```
public string ExecuteScript(string script)
{
     var engine = this.ScriptRuntime.GetEngine("py");

     MemoryStream stream = new MemoryStream();
     this.ScriptRuntime.IO.SetOutput(stream, new StreamWriter(stream));

     engine.CreateScriptSourceFromString(script,
          SourceCodeKind.Statements).Execute(this.ScriptScope);

     int length = (int)stream.Length;
     var bytes = new byte[length];

     stream.Seek(0, SeekOrigin.Begin);
     stream.Read(bytes, 0, (int)stream.Length);

     stream.SetLength(0);
     return Encoding.UTF8.GetString(bytes, 0, length);
}
```

Code Snippet
Scripting anwenden

```
public void ExecuteScript()
{
     var context = this.CreateContextAndModel();
     this.GenerateTestData(context);
     context.ExecuteScript(@"
for projectId in range(10):
  entityObject = Context.CreateProject()
  entityObject.ProjectCode = ""Project "" + str(projectId)
  entityObject.Budget = projectId * 5
  Context.SaveObject(entityObject)

entity = ModelEntity()
entity.Name = ""NewEntity""
prop = TextProperty()
prop.Name = ""NewProp""
entity.Properties.Add(prop)
Context.AddModelEntity(entity)
[…]
     ");
}
```

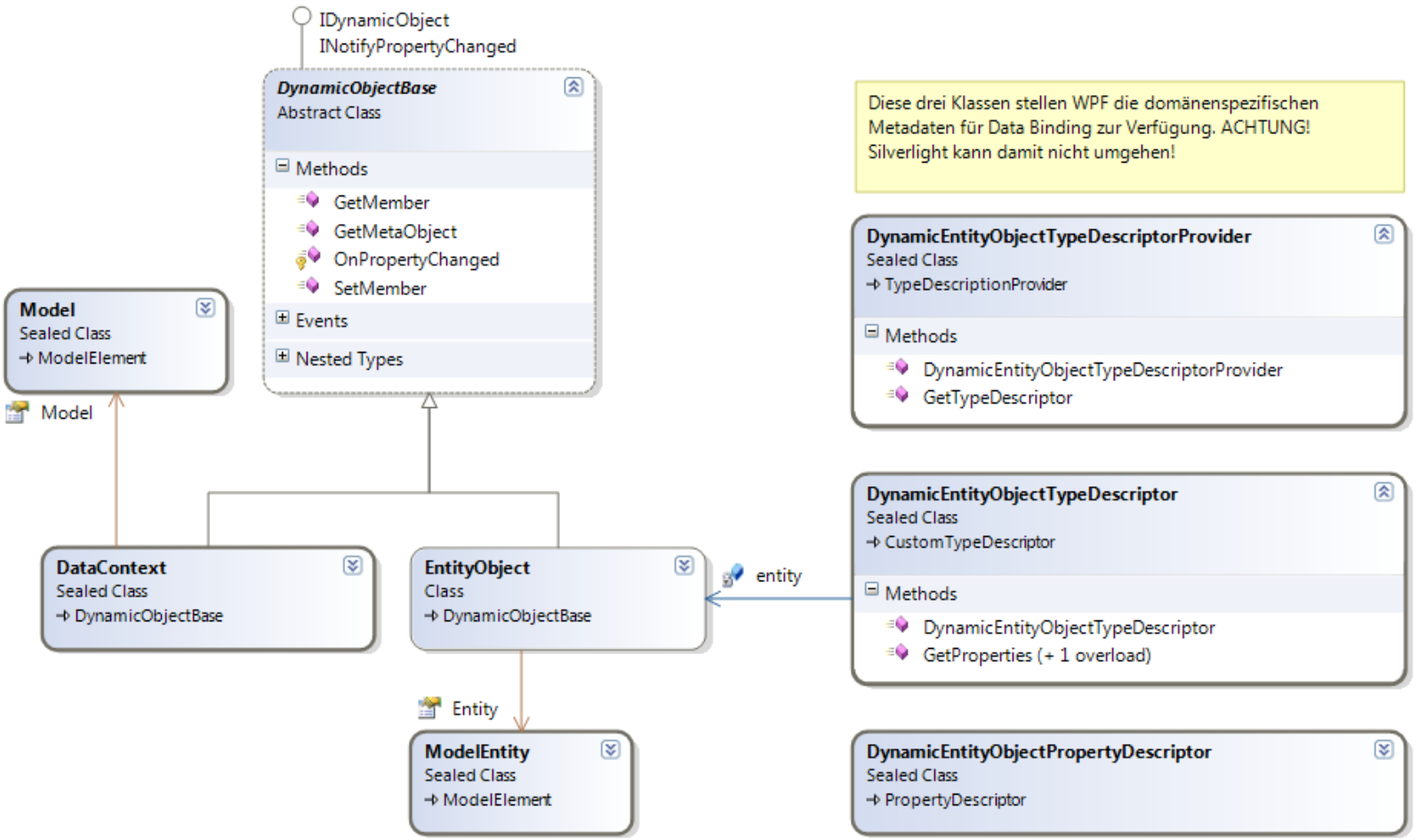
Tipps für Scripting

- Keine eigenen Sprachen erfinden
  - Bestehende (Open Source oder geschützte) verwenden
- Nicht nur für Automatisierung anwendbar
  - Formelsprache der Scripting-Umgebung kann für Erweiterbarkeit genutzt werden
- Achtung: Gefahr von Code Injection bei SaaS → AST vor Ausführung prüfen

Demo 5

Benutzerschnittstelle – alles greift in sich zusammen – hoffentlich ;-)

UML Klassendiagramm
Datenbindung in UI unterstützen



Generelle Tipps zum Abschluss

- Nur dort dynamisch sein, wo es wirklich notwendig ist
- So statisch wie möglich, so dynamisch wie notwendig
- Nutzen fertiger Komponenten wenn sinnvoll
- Keine eigenen Sprachen erfinden…
  - …wenn das nicht Ihr Kerngeschäft ist
- Auf Rahmenbedingungen im Umfeld von SaaS achten
  - Skalierbarkeit
  - Sicherheit
  - Multi-Tenancy

14.–17. 09. 2009
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Vielen Dank!

Rainer Stropek

r.stropek@cubido.at

cubido business solutions gmbh