

NHibernate

Olaf Lischke

Trainer - Berater - Architekt

Multi Tier Applikation

GUI

Präsentation

Windows Forms
WPF
ASP.NET
HTML

- Darstellung von Daten
- UI-Logik (Code Behind)

BL

Logik

*.cs

- Business-/Betriebs-Logik
- Bereitstellung von Daten für die GUI

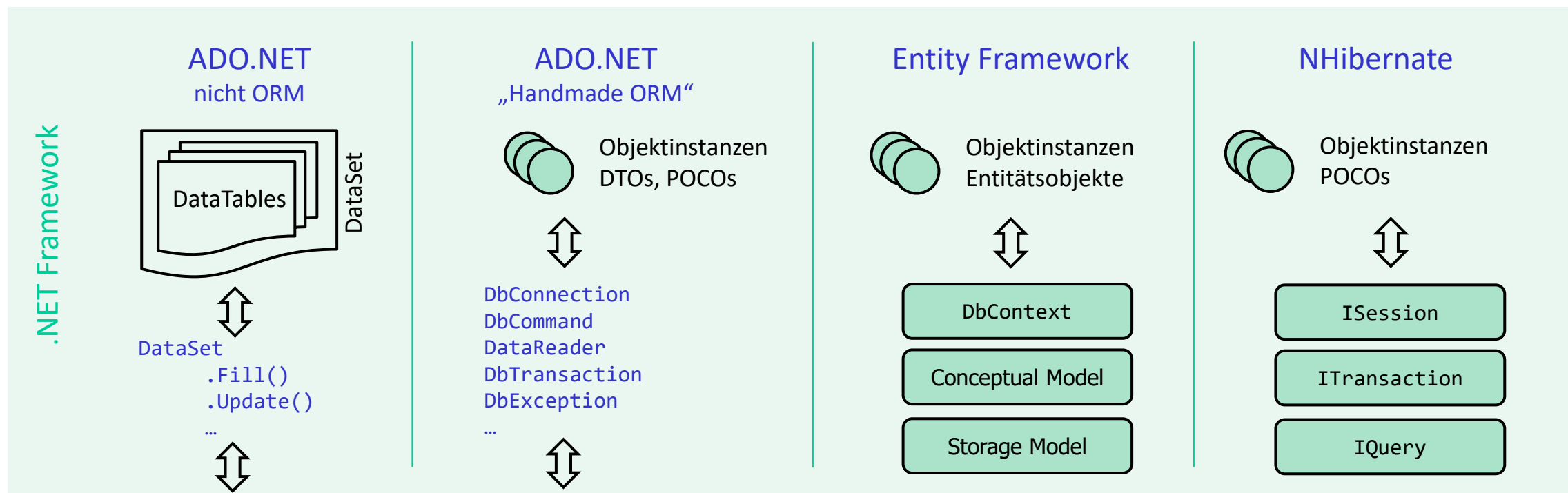
DAL

Persistenz

ADO.NET plain
ORM

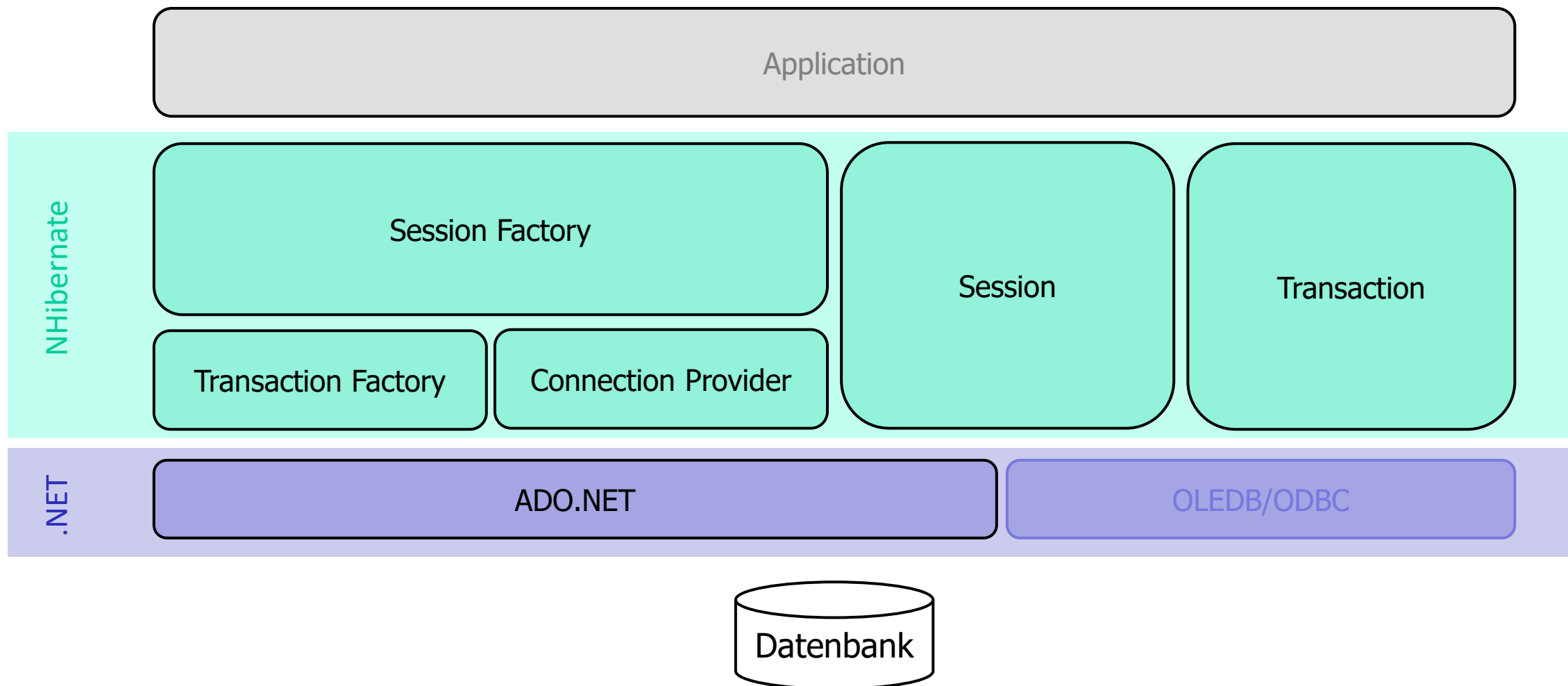
- Aufbewahrung der Daten
- Zurverfügungstellung von Daten

Objekt-Relationales Mapping



- Installationspaket
 - <https://sourceforge.net/projects/nhibernate/files/NHibernate/>
 - inkl. einiger nützlicher Dateien (Schemas, Binaries, ...)
- NuGet-Paket
 - via NuGet Package Manager
- Source Code (Core)
 - <https://github.com/nhibernate/nhibernate-core>
- Doku
 - <https://nhibernate.info/doc/index.html>

NHibernate Architektur



Langlebig

- **ISessionFactory**
 - Immutable
 - Threadsafe
 - Stellt **ISession**-Instanzen bereit

Kurzlebig

- **ISession**
 - Wrapper für **DbConnection**
 - Factory für **ITransaction**
 - SingleThread
- **ITransaction**
 - Abstraktion der **DbTransaction**
 - SingleThread

- Code-basiert

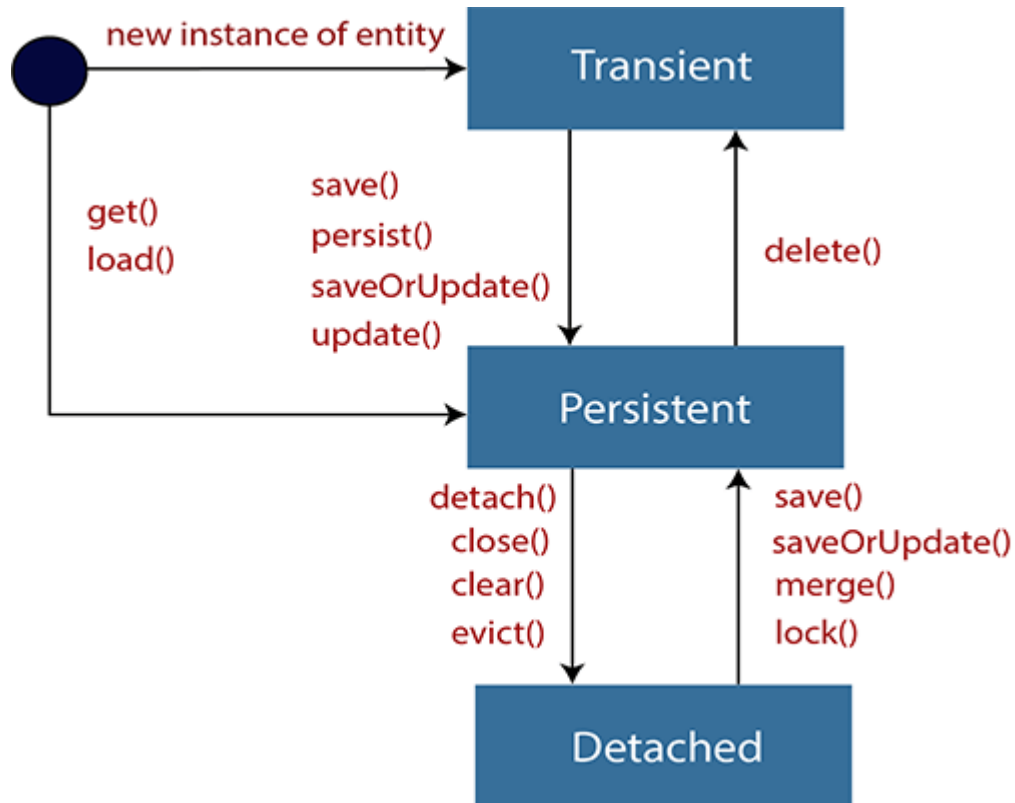
```
cfg.DatabaseIntegration(x => {  
    x.ConnectionString = connectionString;  
    x.Driver<SqlClientDriver>();  
    x.Dialect<MsSql2012Dialect>();  
    x.LogSqlInConsole = true;  
});  
  
cfg.AddAssembly("DemoAssembly");
```

- Per XML

- hibernate.cfg.xml
- app/web.config

```
<hibernate-configuration xmlns = "urn:hibernate-configuration-2.2">  
    <session-factory>  
        <property name = "connection.connection_string">  
            Data Source = .\\sqlexpress;  
            Initial Catalog = DemoDB;  
            Integrated Security = True;  
            Connect Timeout = 15;  
            ApplicationIntent = ReadWrite;  
            MultiSubnetFailover = False;  
        </property>  
        <property name = "connection.driver_class">  
            NHibernate.Driver.SqlClientDriver  
        </property>  
        <property name = "dialect">  
            NHibernate.Dialect.MsSql2012Dialect  
        </property>  
        <mapping assembly = "DemoAssembly"/>  
    </session-factory>  
</hibernate-configuration>
```

Objekt-Lebenszyklus



Transient: POCO ohne Entsprechung in der Datenbank.

Persistent: POCO mit ID, mit einer Session assoziiert. Getrackt durch die Session.

Detached: POCO mit ID, die einem Datensatz entspricht. Ungetrackt.

Quelle <https://www.javatpoint.com/hibernate-lifecycle>

Load() - Lazy

- Erzeugt uninitialisierten Proxy mit Identifier
- `ObjectNotFoundException`
- Bei Zugriff auf Properties (read/write) werden Daten geladen
- One-to-Many/Many-to-One:
Wie oben, jedoch Many-Seite abhängig von Fetching-Strategie (Default: Lazy, nicht geladen).
- Verwaltung durch Session und Cache

Get() - Eager

- Liefert eine Instanz der Entität mit Daten aus der Datenbank – kein Proxy!
- `null`, wenn nichts gefunden
- One-to-Many/Many-to-One:
Alle(!) Daten werden geladen.
Möglich jedoch: Single-Entity mit Get(), abhängige Entität(en) per Load().
- Verwaltung durch Session

Konzept

```
public class NHibernateProxy : MyEntity
{
    private MyEntity target;

    public string FirstName() {
        get {
            if (target == null) {
                target = readFromDatabase();
            }
            return target.FirstName;
        }
    }
    set {...}
}
```

Wann werden Daten gelesen?

- Datenhandling
 - Immediate Fetching
 - Assoziierte werden mitgeladen
 - Lazy Collection Fetching
 - Nur bei Zugriff Collection laden
 - Proxy Fetching
 - Proxy mit Identifier, Daten erst laden bei Zugriff auf Properties

Wie werden Daten gelesen?

- Wieviele/welche SQL Statements
 - Join Fetching
 - Ein Statement mit Outer Join
 - Select/ SubSelect Fetching
 - Zweites Statement für Assoziierte, nur bei tatsächlichem Zugriff
 - „Extra-Lazy“ Collection Fetching
 - Nur verwendete Assoziierte laden
 - Batch Fetching
 - Spez. Select, ein Statement

- *.hbm.xml
 - <hibernate-mapping> - Root, Namespaces, globale Schalter
 - <class> - Definition einer Klasse
 - <name> - Name der Klasse
 - <id> - Primary Key
 - <batch-size> - Anzahl Datensätze für SQL Statement
 - <lazy> - Fetching-Strategie
 - <property> - Definition einer Property
 - <name> - Name der Property
 - <column> - Name der Datenbankspalte

- `<Many-to-One>` - assoziiert zu anderer Instanz
 - `<name>` - Propertyname
 - `<column>` - Spaltenname in Datenbank
 - `<class>` - Klassenname
 - `<cascade>` - Verhalten bei CRUD-Operationen
 - `<fetch>` - Fetching-Strategie
 - `<lazy>` - Proxy-Verhalten
 - `<inverse>` - Bidirectionale Assoziation
- `<set>`, `<list>`, `<map>`, `<bag>`, `<array>` - Collection Typ
- `<One-to-Many>` - assoziiert zu Collection

HBML	Bemerkungen	.NET
<list>	Sortierte Menge von Elementen, nicht zwingend unique.	<code>IList<T></code>
<set>	Unsortierte Menge an unique Elementen. Exception beim Einfügen doppelter Einträge.	<code>HashSet<T></code>
<bag>	Unsortierte Menge an Elementen, nicht zwingend unique.	<code>IList<T></code>

- **Table per Class Hierarchy**

`<subclass name=".." discriminator-value="..">`

- Alles in einer Tabelle
- NULL-Spalten für Properties aus abgeleiteten Klassen
- Discriminator zur Typ-Identifizierung

- **Table per Subclass**

`<joined-subclass name=".." table="..">`

- 1 Tabelle für Basis, 1 Tabelle je abgeleiteter Klasse

- **Table per Concrete Class**

`<union-subclass name=".." table="..">`

- 1 Tabelle je abgeleiteter Klasse
- Redundante Spalten für Properties aus Basisklasse

- **HQL** – **H**ibernate **Q**uery **L**anguage
- **Criteria** Queries
- **QueryOver** Queries
- **LINQ** – **L**anguage **I**Ntegrated **Q**uery
- Native SQL

HQL – Hibernate Query Language

- Alt, aber bewährt
- SQL-artig, aber objektbasiert

```
var q = session.CreateQuery("select tr from Track tr  
                             where tr.Name like 's%' and tr.Genre.Name like 'Metal'");
```

Diagram annotations:

- IQuery** (blue text) with an arrow pointing to `q`.
- Property** (blue text) with an arrow pointing to `tr.Name`.
- Type** (blue text) with an arrow pointing to `Track`.

- Methodenorientiert
- Implementierung des Query-Objekts

ICriteria
↓

```
var q = session.CreateCriteria<Track>()  
    .Add(Expression.Like("Name", "S%"))  
    .CreateCriteria("Genre")  
        .Add(Expression.Like("Name", "Metal"));
```


Property

- ExtensionMethods und Lamda-Ausdrücke
- Statische, typsichere Wrapper für **ICriteria**

IQueryOver
↓
`var q = session.QueryOver<Track>()
 .Where(tr => tr.Name.IsLike("S%"))
 .JoinQueryOver<Genre>(tr => tr.Genre)
 .Where(g => g.Name == "Metal");`

- Linq-to-Hibernate Provider
- IQueryable

IQueryable<T>



```
var q = session.Query<Track>()  
                .Where(tr => tr.Name.StartsWith("S")  
                        && tr.Genre.Name == "Metal");
```

Native SQL

- SQL-Statements zur direkten Ausführung
- Auch Updates/Stored Procedures/Functions
- Können als Named Query per XML deklariert werden

ISQLQuery

```
var q = session.CreateSQLQuery("SELECT * FROM Track AS tr" +  
                                "INNER JOIN Genre AS g ON tr.GenreId = g.GenreId" +  
                                "WHERE tr.Name LIKE 'S%' AND g.Name LIKE 'Metal'" +  
                                )  
                                .AddEntity(typeof(Track));
```

↑
definiert den Rückgabety, sonst `object[]`

Session.

- Save(Instanz)
 - Änderungen/neues Objekt werden geschrieben
- Update(Instanz)
 - Änderungen werden geschrieben
- Evict(Instanz)
 - Objekt wird 'detached'
- Delete(Instanz)
 - Objekt wird 'transient' (und seine Daten aus der DB gelöscht), existiert weiterhin im Speicher

- Synchronisierung Speicherdaten mit Datenbank
- Explizit aufgerufen durch
 - Session.Flush(), Transaction.Commit()
- Implizit bei Query.List(), Query.Enumerable() u.a.
- Ablauf
 - Entity Inserts
 - Entity Updates
 - Collections Deletes
 - Collections Elements Deletes
 - Collections Inserts
 - Entity Deletes

Except when you explicitly `Flush()` there are absolutely no guarantees about *when* the Session executes the ADO.NET calls, only the *order* in which they are executed.
Quelle: <https://hibernate.info/doc/hibernate-reference/manipulatingdata.html>