Neo

.NET Entity Objects

http://neo.sourceforge.net

Architecture and Implementation

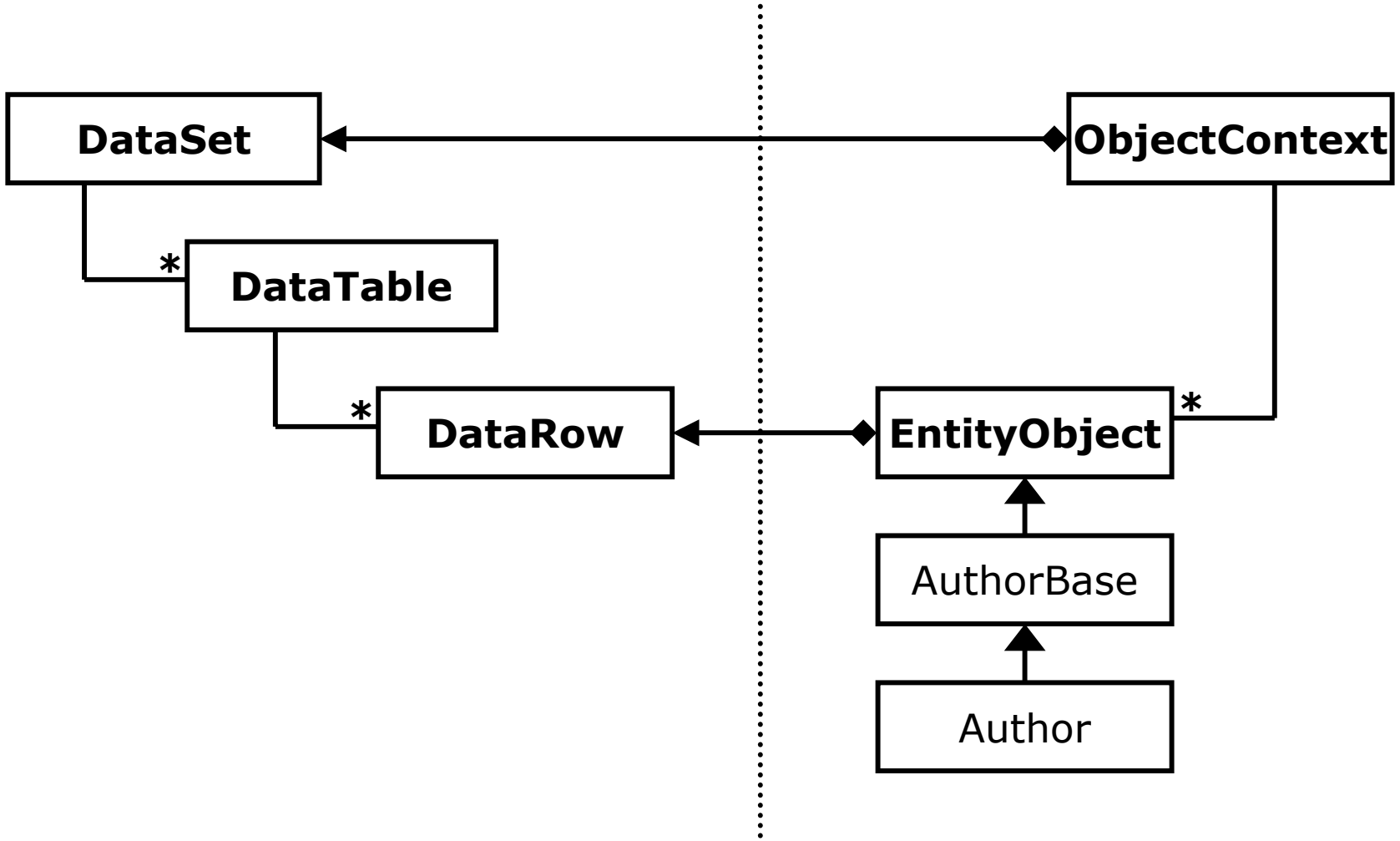
Architecture and Implementation 1

- The Big Picture
- System.Data integration
- Overview of the main classes

- More gory details
- What if I like the JDO approach?

Architecture: The Big Picture

The Entity Object world and the MS.NET world are brought together by **Composition**



Impl: System.Data integration 3

Every entity object has a reference to its DataRow and to the context it belongs to.

```
public class EntityObject {
  public DataRow Row { get { … }; }
  public ObjectContext Context { get { … }; }
```

Data is stored in directly in row:

```
public string FirstName {
  set { Row["au_fname"] = value; }
  get { return Row["au_fname"]; }
```

Note how property names and column names can be mapped.

Impl: System.Data integration 4

To-one relation properties take the value from the related object and set it as foreign key value

```
public class Title {
  public Publisher Publisher {
    set {
      Row["pub_id"] = value.Row["pub_id"];
    }
```

They use the context to find the object for the foreign key value in the related table

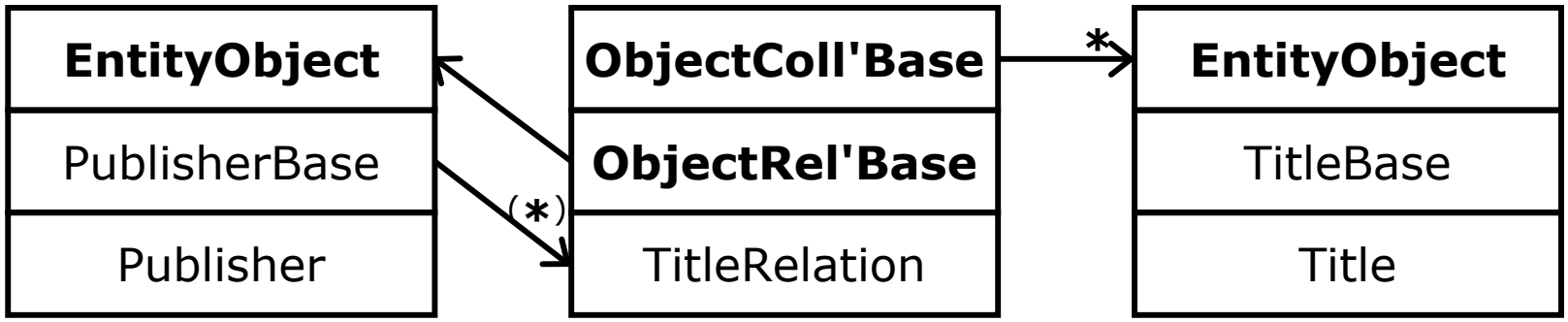
```
    get {
      return (Publisher)Context.
        GetObject("publishers", Row["pub_id"]);
    }
```

Impl: System.Data integration

To-many relations are managed by and exposed as typed collections

```
public class Publisher {
  public readonly TitleRelation Titles;
```

The relation collection objects have a reference to the object that owns the relation and a cache for the objects in the relation



Impl: System.Data integration

Relation objects work much like the relation properties. They manipulate the foreign key column of the affected object.

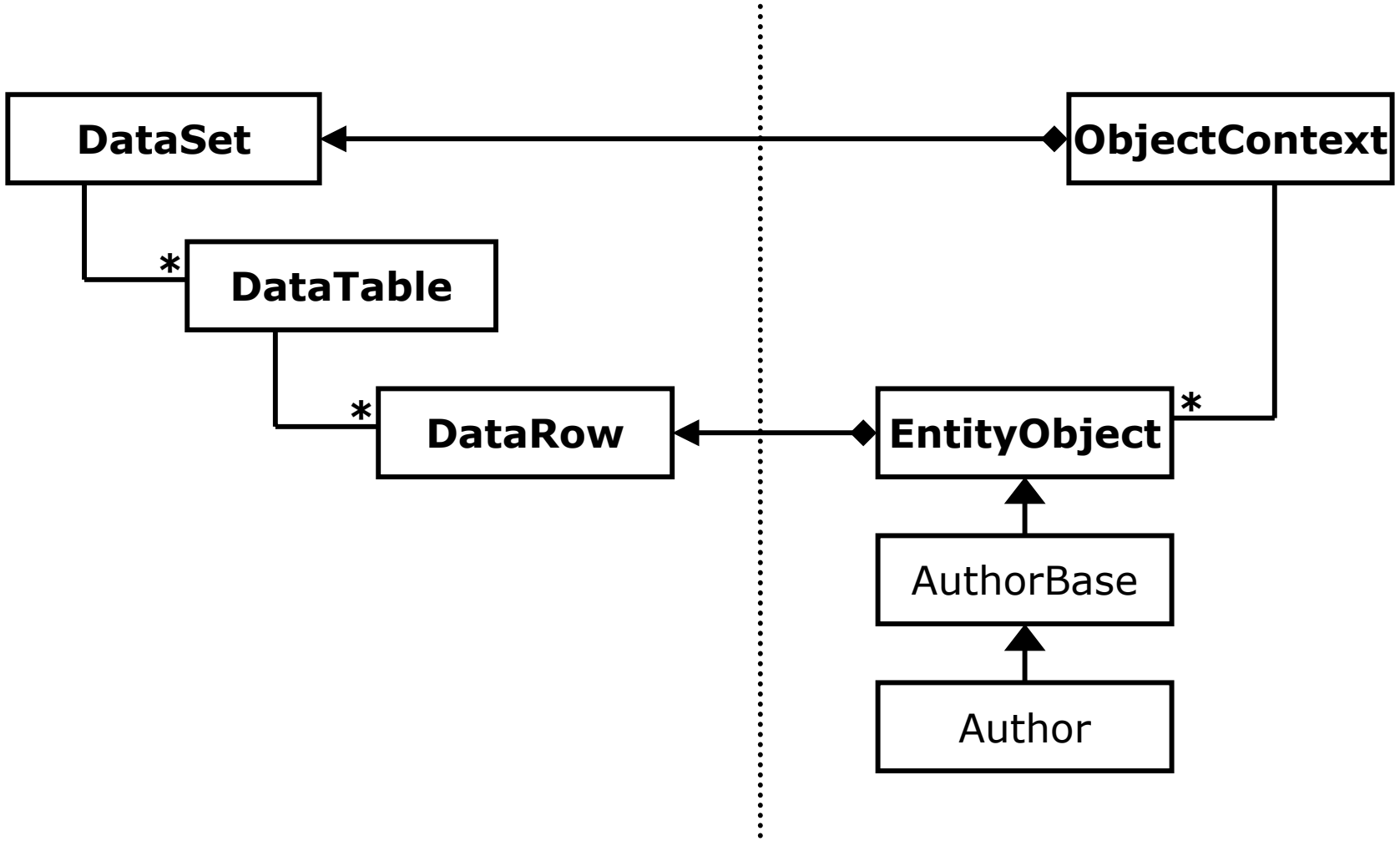
```
public void Add(Title aTitle) {
  aTitle.Row["pub_id"] = Owner.Row["pub_id"];
}
public void Remove(Title aTitle) {
  aTitle.Row["pub_id"] = null;
}
```

and use the context to find objects

```
protected void Load() {
  innerList = Owner.Context.GetObjects( … );
```

Architecture: The Big Picture

The Entity Object world and the MS.NET world are brought together by **Composition**



Architecture: The Object Context

Applications can use a shared context (one per execution thread) or create their own contexts.

```
context = ObjectContext.SharedInstance;
context = new ObjectContext();
```

ObjectContext actually handles the CRUD operations, other methods are just a façade.

```
public Class ObjectContext {
 IEntityObject CreateObject(Type objectType);
 IList GetAllObjects(Type objectType);
 IList GetObjects( … );
 void DeleteObject(object anObject);
```

Architecture: The Object Context

The context relies on its internal DataSet to track changes (creates, updates and deletes.) It provides methods to save or reject the changes.

```
public ICollection SaveChanges();
public void RejectChanges();
```

The DataSet property is writable!

```
context.DataSet = anotherDataset;
```

Factories create entity objects in a specific context. They either use the shared context or a specific context passed to the constructor.

```
authorFactory = new AuthorFactory(myContext)
```

Architecture: Data Stores

A data store is responsible for retrieving and updating the backing store for the context.

```
public interface IDataStore {
   DataTable Fetch(IFetchSpecification fs)
   ICollection Save(DataTable table)
   void BeginTransaction()
   …
```

ObjectContext uses a data store to retrieve objects and save changes.

A special data store could act as a façade for other data stores. I could select an appropriate store for a request and forward the request.

Architecture: Disconnected clients

Object contexts can work without a data store and take data directly from a DataSet.

```
objects = context.RegisterObjectsForDataset(ds)
```

They allow indirect access to their internal DataSet so that changes can be made persistent.

```
ds = context.GetChanges()
// do something to persist these changes,
// maybe send dataset to a server...
if(success)
  context.AcceptChanges()
else
  context.RejectChanges()
```

Architecture: PkChangeTable

Used to redistribute database generated keys.

Neo generates temporary (negative) primary key values. Data stores must provide a mapping from these temporary keys to the actual db values.

```
table = new PkChangeTable("jobs");
table.AddPkChange(row["job_id"], actualKeyVal);
```

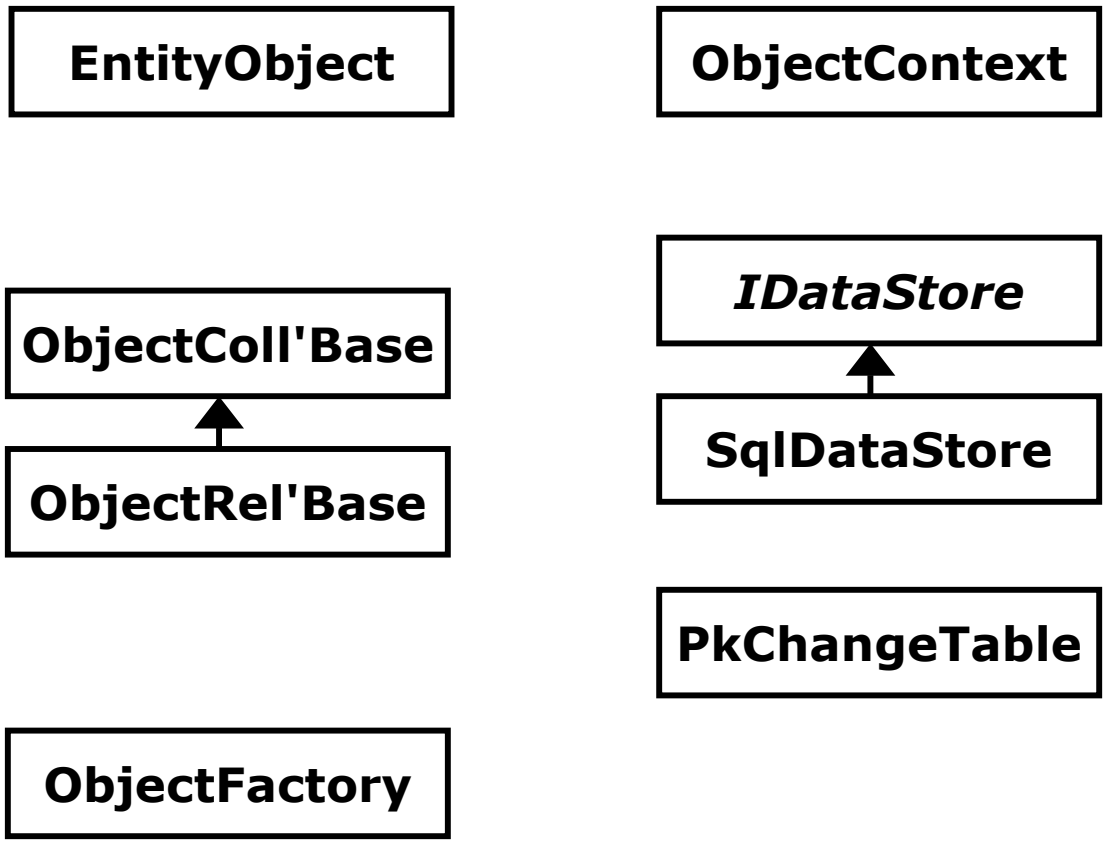
Use in distributed environments:

```
ds = clientContext.GetChanges();
```

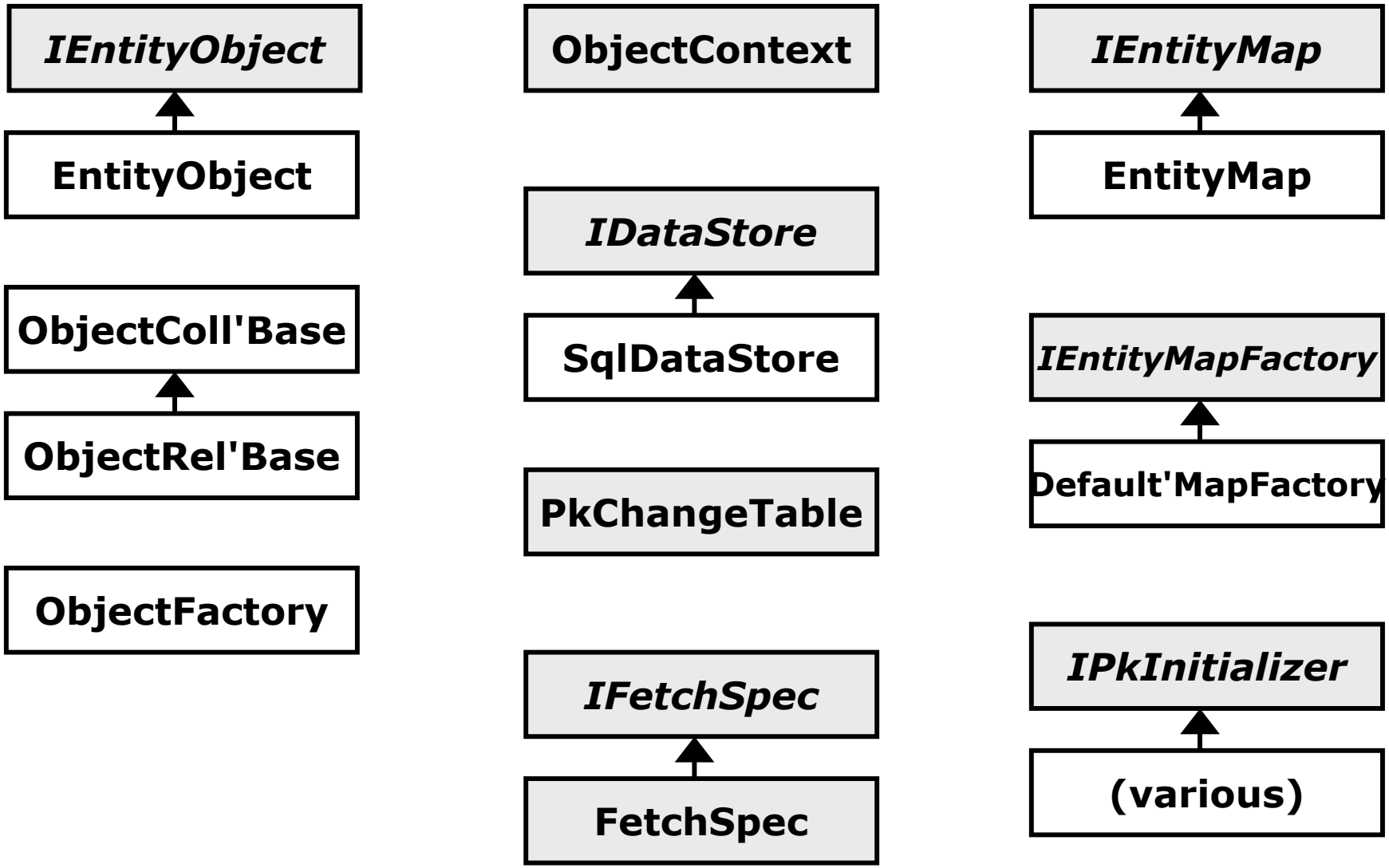
```
serverContext.RegisterObjectsForDataSet(ds)
pkctArray = serverContext.SaveChanges();
```

```
clientContext.UpdatePrimaryKeys(pkctArray);
clientContext.AcceptChanges();
```

Architecture: Main classes



Architecture: Class overview



Neo.Core and IEntityObject

IEntityObject is the minimal interface for an entity object:

```
public interface IEntityObject {
  Neo.Core.ObjectContext Context { get; }
  System.Data.DataRow Row { get; }
}
```

In addition, entity objects must this constructor:

```
public SomeClass(DataRow row, ObjectContect ctx)
```

The core classes use IEntityObject only and do not impose the EntityObject/Base/User Class structure. Consequently, Neo.Core can be used with other entity object models.

Internals: IFetchSpecification

IFetchSpecification is used to select objects in object contexts and data stores.

```
public interface IFetchSpecification {
  public IEntityMap EntityMap { get; }
  public Qualifier Qualifier { get; }
  }
```

FetchSpecification is a straight-forward impl

```
public class FetchSpecification : IFetchSpec… {
  public FetchSpecification(IEntityMap m, …)
  }
```

Query Templates implement IFetchSpecification.

Internals: IEntityMap

Provides information about the mapping between database tables and entity objects.

```
public interface IEntityMap {
  Type ObjectType { get; }
  string TableName { get ; }
  string[] Columns { get; }
  string GetColumnForProperty(string prop);
```

EntityMap implementers must also be able to generate System.Data schema information.

```
  void UpdateSchema(DataTable table, …)
```

Neo provides an abstract implementation that is subclassed by generated classes, one per entity.

Internals: IEntityMapFactory

Entity map factories provide access to entity maps by table name or object type.

```
public interface IEntityMapFactory {
  IEntityMap GetMap(string tableName);
  IEntityMap GetMap(Type objectType);
  ICollection GetAllMaps();
}
```

DefaultEntityMapFactory is standard implementation that searches for IEntityMap implementers in all linked assemblies.

Internals: IPkInitializer

Contains a single method which must initialise a new database row. If scheme involves pk values provided by the application they are passed in.

```
public interface IPkInitializer {
  void InitializeRow(DataRow row, object arg);
}
```

Neo has three implementations.

```
public class UserPkInitializer

public class NativePkInitializer

public class GuidPkInitializer
```

Architecture: Class overview