Neo

.NET Entity Objects

http://neo.sourceforge.net

Design Goals

Vision and Design Goals

- Business entities are represented by objects
- One Entity Object class per database table
- One Entity Object per database row
- Transparent access to derived properties
- Transparent access to related entities
- Strongly typed API
- Automatic generation of database schema and class templates from a single XML file
- Separation of generated and custom code
- Full integration with System.Data framework

Design: Properties

Entity object has properties for db columns.

```
public class Author {
  public string FirstName {
    get { … }
    set { … }
  }
```

Usage:

```
Console.WriteLine(anAuthor.FirstName);
anAuthor.FirstName = "Haruki";
```

Design: Derived Properties

Custom code adds new (derived) properties.

```
public class Author {
  public string FullName {
    get { return FirstName + " " + LastName; }
    set { … }
  }
```

Usage: (indistinguishable from db properties)

```
Console.WriteLine(anAuthor.FullName);
anAuthor.FullName = "Haruki Murakami";
```

Design: To-One Relations

Entity object has properties for related objects.

```
public class Title {
  public Publisher Publisher {
    get { … }
    set { … }
  }
```

Usage: (simple object assignments)

```
thePublisher = aTitle.Publisher;
Console.WriteLine(thePublisher.Name);
aTitle.Publisher = anotherPublisher;
```

Design: To-Many Relations

Object has (typed) collection for related objects.

```
public class Publisher {
  public TitleRelation Titles { get { … }; }

public class TitleRelation : IList {
  public Title this[int index] { … }
  public void Add(Title aTitle) { … }
  public int Count { … }
```

Usage: (identical to .NET collection pattern)

```
if(thePublisher.Titles.Count > 0)
   aTitle = thePublisher.Titles[0];
thePublisher.Titles.Add(anotherTitle);
```

Objects are loaded on first access: Lazy-Loading

Design: CRUD, the rest

Factories create and find objects. Objects delete themselves

```
public class AuthorFactory {
  public Author CreateObject()
  public AuthorList FindAllObjects()
  public AuthorList FindMatchingObjects( … )
  public AuthorList Find( … )

public class Author {
  public Delete()
```

Usage:

```
myFactory = new AuthorFactory();
newAuthor = myFactory.Create();
resultSet = myFactory.FindAllObjects();
```

Design: Primary Keys

Primary keys can be generated; either auto-incremented integers or globally unique ids.

Neo also supports "meaningful" primary keys and generates different Create methods

```
title = titleFactory.CreateObject("TC7777");
```

Neo understands correlation tables and generates special Create methods

```
ta = taFactory.CreateObject(title, author);
```

Different primary key generation strategies can be implemented outside the framework.

Design: Query Templates

Factories can create a query template that has the same properties as the corresponding entity object, including to-one relationships.

```
template = authorFactory.GetQueryTemplate();
template.FirstName = "Haruki"

template = titleFactory.GetQueryTemplate();
template.Publisher = aPublisher;
```

Factories can find all objects matching the template

```
titles = factory.FindMatchingObjects(template);
```

Design: Qualifiers

Qualifiers define criteria for selections. They are normally constructed using formats.

```
q = Qualifier.Format("name = {0}", input);
```

Formats can use inlined values and comprise multiple clauses:

```
q = Qualifier.Format("name = 'Haruki');
q = … ("name = {0} and locked = false", input);
```

Formats provide a shortcut for simple matches:

```
q = new Qualifier.Format("Name", input);
```

Design: Qualifiers

Qualifiers can evaluate whether an object matches their criteria:

```
if(q.EvaluateWithObject(anAuthor))
    doSomething(anAuthor);
```

Factories use qualifiers:

```
AuthorFactory f = new AuthorFactory();
AuthorList list = f.Find("Name = {0}", input);
```

So do collections and relations:

```
sublist = list.Find("Advance < 5000");
sublist = publisher.Titles.Find( … );
```

Design: Delete modes

SetNull deletes behave as expected

```
aTitle = thePublisher.Titles[0];
thePublisher.Delete()
if(aTitle.Publisher == null)
  Console.WriteLine("It just works!");
```

Cascading deletes are supported and the last line in the following example will cause an exception to be thrown

```
aTitle = theAuthor.Titles[0];
theAuthor.Delete()
Console.WriteLine(aTitle.PublicationDate);
```

All methods presented are implemented in an intermediary class, AuthorBase for example. This should not be modified.

```
public class AuthorBase : EntityObject {
   public string FirstName { … }
   public string LastName { … }
```

Initially, a subclass is generated but the developer is expected to modify it and it is not re-generated when the schema changes.

```
public class Author : AuthorBase {
   public string FullName { … }
```

Design Goals (review)

- ✓ Business entities are represented by objects
- ✓ One Entity Object class per database table
- ✓ One Entity Object per database row
- ✓ Transparent access to derived properties
- ✓ Transparent access to related entities
- ✓ Strongly typed API
- Automatic generation of database schema and class templates from a single XML file
- ✓ Separation of generated and custom code
- Full integration with System.Data framework