AndyPatterns

Relationship Manager for Boo

Here is an implementation of RM Relationship Manager for .NET using the Boo language (porting it from Python). The resulting assembly is usable from C# and other .NET languages. See C# example below. If you are interested in the pure .Net 2.0 C# port of relationship manager, click here.

P.S. Boo is a .NET language - a sort of Python blended with C# creating an interesting hybrid - to ready my thoughts about on Boo (and the potential synthesis of static and dynamic typing) click here. Here are my tips for debugging in Boo, which are likely completely out of date now.

Installation

Download the ready to use compiled **assembly** (includes a sample exe)

To relationship manager in your own projects just add the reference to the assembly **RelationshipManager55.dll** and create a RM1 instance and then make calls on it using the API below. e.g. R(), NR() etc. A full example is shown below.

Source code

View the Boo source code of Relationship Manager for .NET online (color coded) on the Boo wiki page. If you want to compile it yourself then download the entire SharpDevelop project here.

Documentation

The API of the relationship manager for .NET is described in RM Usage and Theory. Note the Boo .NET assembly only uses the shorthand names e.g. ER() instead of EnforceRelationship(). I'll fix this one day. Also note that BS(t) is not implemented in the boo port. This is a temporary omission. If you need this method urgently, edit RM.boo and add the following to the class RM1:

```
def BS(toObj, relId):
   # findObjectsPointingToMe(toMe, id, cast)
   return self.rm.FindObjects(null, toObj, relId)
```

Basic API

Returns	Function Name	Short-hand
void	addRelationship(f, t, id)	R(f,t)
void	removeRelationship(f, t, id)	NR(f,t)
Vector	findObjectsPointedToByMe(f, id)	PS(f)
Vector	findObjectsPointingToMe(t, id)	BS(t)

Extra API

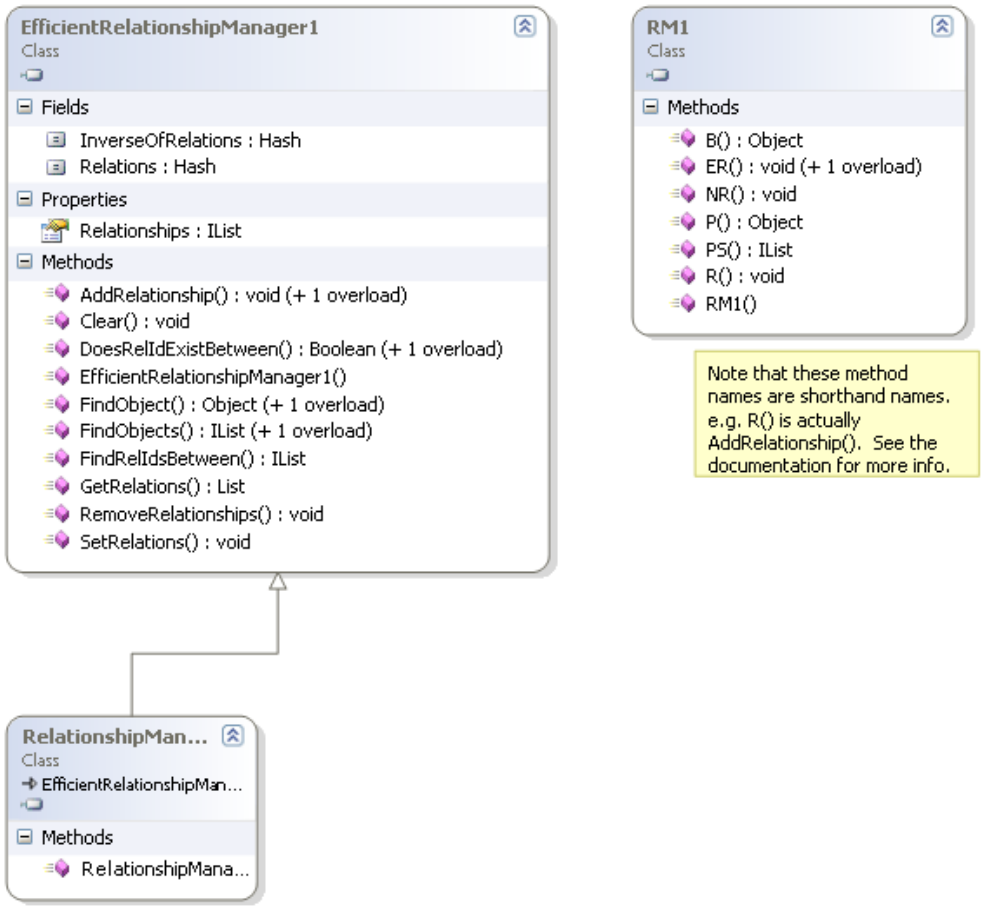
Returns	Function Name	Short-hand
void	EnforceRelationship(id, cardinality, bidirectionality)	ER(id, c, bi)
Object	findObjectPointedToByMe(fromMe, id, cast)	P(f)
Object	findObjectPointingToMe(toMe, id cast)	B(t)
void	removeAllRelationshipsInvolving(o, id)	NRS(o)

The extra API allows you to enforce relationships e.g.

```
ER("xtoy", "onetoone", "directional")
```

registers the relationship as being one to many and directional, so that e.g. when you add a second relationship between the same two objects the first relationship is automatically removed - ensuring the relationship is always one to one. Alternatively, you could raise and exception.

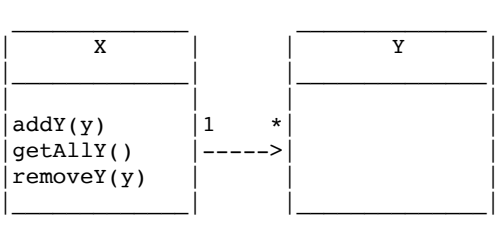
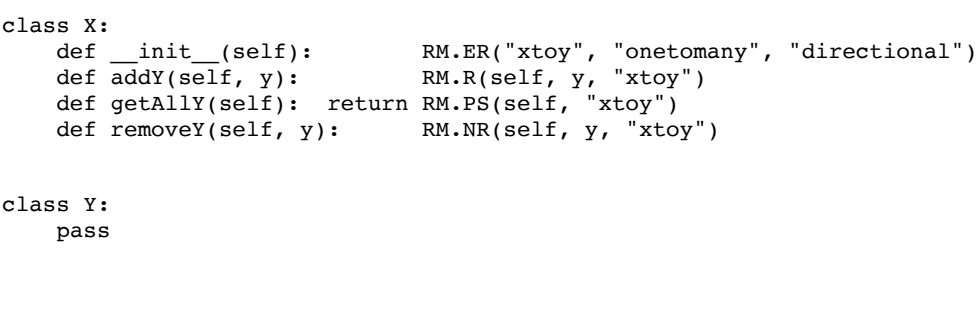
The extra API also adds a pair of find methods that only find *one* object, and cast it to the appropriate type. This is a commonly used convenience method.



UML class diagram for the Boo .NET relationship manager implementation.

What methods do I put where when modelling relationships?

I recommend that you use the templates when figuring out what methods to put where for each type of classic relationship you want to model. For example, to implement a **one to many** relationship between two classes X and Y, you might use the following set of methods:

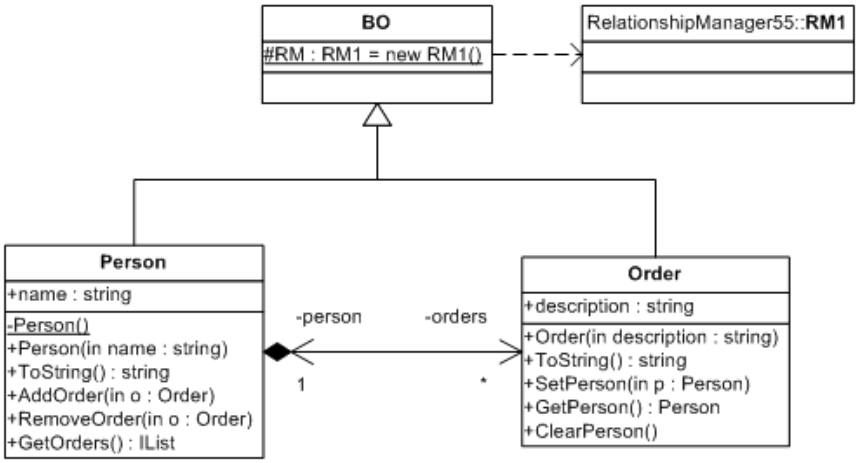
one to many	
1 --> * *directional*	
Plural API No API  	picture
 	template

Thus if you have a class called **Customer** and a class called **Order**, then the Customer class would correspond to class X and the Order class to class Y. Your methods might be AddOrder(o) which corresponds in the above example to addY(y).

Its up to you to rename both the class and method names of the template to match your own classes.

A concrete example

Say you want to model a Person class which has one or more Orders. The Orders class has a backpointer back to the Person owning it.



Instead of hand coding and reinventing techniques for doing all the AddOrder() methods and GetOrders() methods etc. using ArrayLists and whatever, we can do it using the relationship manager object instead, which turns out to be simpler and faster and less error prone.

The RM (relationship manager) is implemented in this particular example as a static member of the base BO (business object) class. Thus in this situation all business objects will be using the same relationship manager.

Here is the c# code to implement the above UML:

```
using System;
using System.Collections;
using RelationshipManager55;

namespace WindowsApplicationUsing_RelationshipManagerDllTest001
{
        ///
        /// BO is the base Business Object class which holds a single static reference
        /// to a relationship manager. This one relationship manager is
        /// used for managing all the relationships between Business Objects.
        ///
        public class BO  // Base business object
        {
                static protected RM1 RM = new RM1();
        }

        ///
        /// Person class points to one or more orders.
        /// Implemented using a relationship manager rather
        /// than via pointers and arraylists etc.
        ///
        public class Person : BO
        {
                public string name;

                static Person()
                {
                        RM.ER("p->o", "onetomany", "bidirectional");
                }

                public Person(string name)
                {
                        this.name = name;
                }
                public override string ToString()
                {
                        return "Person: " + this.name;
```

```
                }

                public void AddOrder(Order o)
                {
                        RM.R(this, o, "p->o");
                }
                public void RemoveOrder(Order o)
                {
                        RM.NR(this, o, "p->o");
                }
                public IList GetOrders()
                {
                        return RM.PS(this, "p->o");
                }
        }

        ///
        /// Order class points back to the person holding the order.
        /// Implemented using a relationship manager rather
             /// than via pointers and arraylists etc.
        ///
        public class Order : BO
        {
                public string description;

                public Order(string description)
                {
                        this.description = description;
                }
                public override string ToString()
                {
                        return "Order Description: " + this.description;
                }

                public void SetPerson(Person p)
                {
                        RM.R(p, this, "p->o");  // though mapping is bidirectional,
                                        there is still a primary relationship direction!


                }
                public Person GetPerson()
                {
                        return (Person) RM.P(this, "p->o");
                }
                public void ClearPerson()
                {
                        RM.NR(this, this.GetPerson(), "p->o");
                }
        }

}
```

Here is the project source code WindowsApplicationUsing RelationshipManagerDllTest001.rar