domain object vs. business object

The terms "business object" and "domain object" are mutually exchangeable in common understanding in enterprise computing. Note that wikipedia links to the same article for both terms:

- http://en.wikipedia.org/wiki/Domain_object
- http://en.wikipedia.org/wiki/Business_object_%28computer_science%29

In re-motion, these terms are related, but not mutually exchangeable. Important differences exist, and this page clarifies these differences and the precise meaning of each term.

Both business objects and domain objects are collections of properties, modelled after the real-world entities you want to manage with your application. For example, if you want to manage people with your re-motion application, you might want *person* objects. A toy person object can consist of these (typed) properties:

- First name = "Sigmund"
- Surname = "Freud"
- Title = "Doctor"
- Birth date = "1856-05-06"

To this vague explanation all enterprise programmers can agree. What about the differences?

In re-motion,

- a *domain object* can persist its value in a database
- a *business object* exposes an `IBusinessObject` interface, so that a binding layer can discover which properties constitute the (business object)

If we understand the Dr. Freud object listed above as a business object, we know that we can use the `IBusinessObject` interface to initiate a dialog like this:

Q: Unknown object, what' your class?
A: `Person`
Q: What properties constitute that class?
A: A first name, a surname, a title and a birth date.
Q: What are the values and types of those?
A: the first name is the string "Sigmund";
the surname is the string "Freud"; the title is the string "Doctor";
the birth-date is the date "1856-05-06".

After such a conversation, re-bind (or some other binding layer, that's the beauty of it) knows everything it needs to display all of Dr. Freud's values in a form or a list. That's the business object aspect of that instance, and it is most important for re-form.

If we understand the Dr. Freud object as a domain object, we can be sure that it can load and store itself from and to some database. That's the domain object aspect of that instance, and it is most important for re-store.

Not all business objects are domain objects

Aren't all domain objects business objects? What's the point of a business object that can't be persisted?

Note that re-store and re-bind are completely independent from each other.

- re-bind might bind to non-re-store objects (such implementations exist at rubicon)
- re-store might be used in a command line tool that has no need for re-bind, because nothing is displayed (such programs do exist mainly for data migration or diagnostics)

"Search objects" expose an `IBusinessObject` interface, but are never persisted. Like domain objects, they are sets of typed properties, but these properties are search terms, not components of objects you want to manage. If you have person objects in your application, it is a good idea to have person search objects as well:

- First Name = whatever
- Surname = "Smith"
- Title = Ph.D.
- birth-date = before 1970-01-01

This search object represents a query that gives you all Ph.D.s named Smith who are born before 1970. It exposes an `IBusinessObject` interface,

- because it is made of typed properties
- `IBusinessObject` is useful for discovering typed properties

Conversely, domain objects ***DON'T*** expose an `IBusinessObject` interface. If you want objects with persistence **AND** an `IBusinessObject` interface, you must resort to ***bindable*** *domain objects*. `BindableDomainObject` is the base class for all re-motion web applications, because they are domain objects (persistence) **AND** business objects (can be displayed by re-bind).

See also

bindable domain object
domain object class
domain assembly
IBusinessObject interface