Organic layered programming

An extension to Domain-Context-Interaction (DCI)

The current OO-programming model is very simple and very class oriented.
The object is simply a container for state and behavior, where all state and behavior is mixed in one big bag. State and behavior can have different security levels (private, protected, public) and can be set to belong to either the class or object. In languages such as Java and C#, you have interfaces, abstract, static and other such low-level modeling abstractions, but no higher level abstractions of note.

The "compile and run" programming model is not suited for a true Object-Oriented programming model, since everything has to be statically defined at runtime. The program thus becomes static, unable to cope with a changing environment.

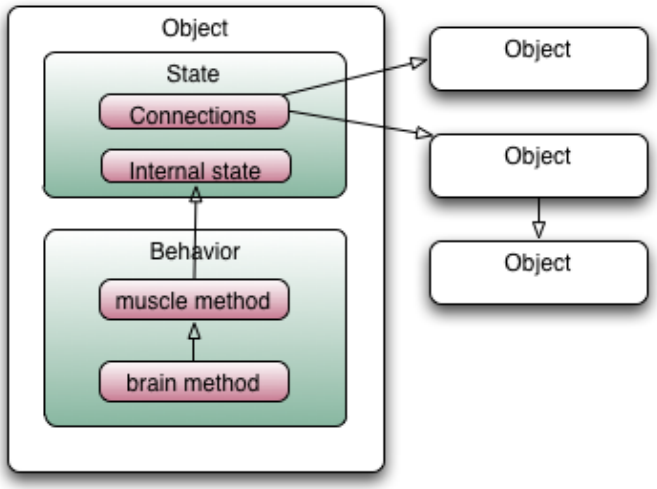

Figure 1 Current Object model

There are many problems inherent in this low-level OO-programming model, considering the complex usage scenarios of today.

DESCRIBE PROBLEMS and LIMITATIONS
- SOLUTIONS

This leads me to propose the following OO-programming model, where an Object is divided into multiple distinct layers according to behavior, and where there are clear enforceable contracts as to the intra-layer communication. Muscle methods only operate on bones and so on.

The nerves layer can be used as a synchronization layer, for ex. to enable asynchronous modes of operation.

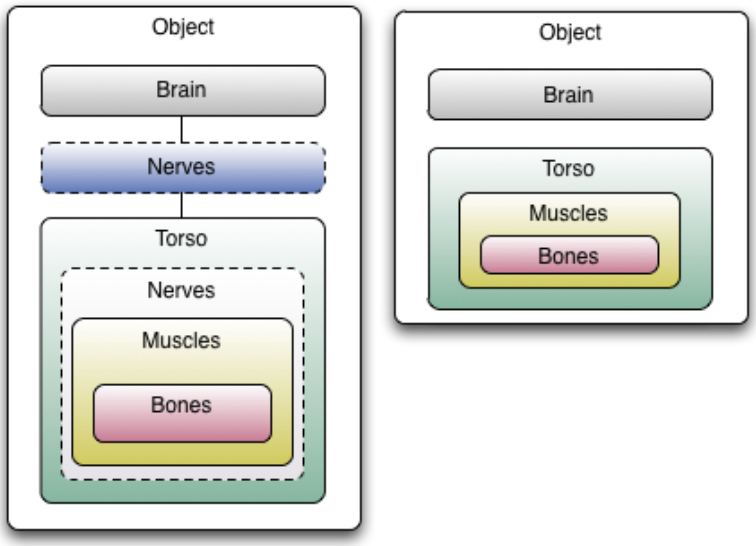

Figure 2 Organic object layers

Figure 2 displays how an object can be logically grouped into various layers similar to the typical layers of an organic organism.

The main brain-ware functions of the object should be placed in the *brain layer*.
The *muscle layer* should only contain methods that operate on the *bones layer* (also known as the object skeleton). The skeleton contains one or more bones. The bones are the private states of the object.

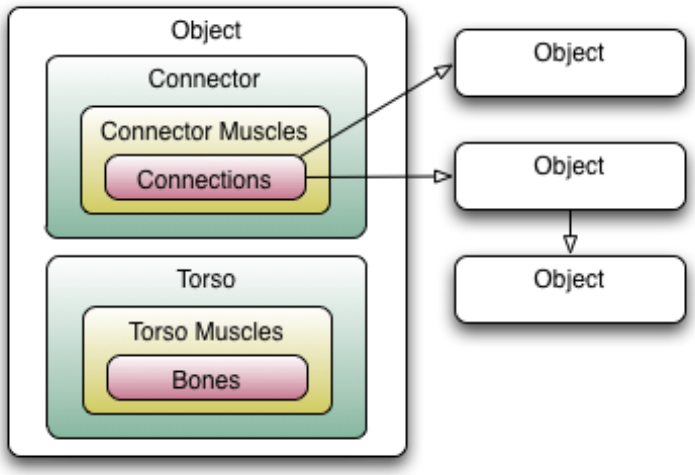

Figure 3 Object Connectors

The object can also be connected to other object through the *connector layer*.
This way we make explicit which state is contained within the object itself, and which state is connections or references to objects outside the object itself.

Objects live in a world of objects at runtime. This should be reflected in the programming model and runtime environment. I propose a World object, which acts as the container for the execution of objects. The World has an Actors container where acting objects can be placed. There is also a Context container where different Context objects are registered.
On different events, an object can change context and the object should transform itself by applying one or more new roles in order to function in the new context.
The application of roles according to context is handled by the World Controller.

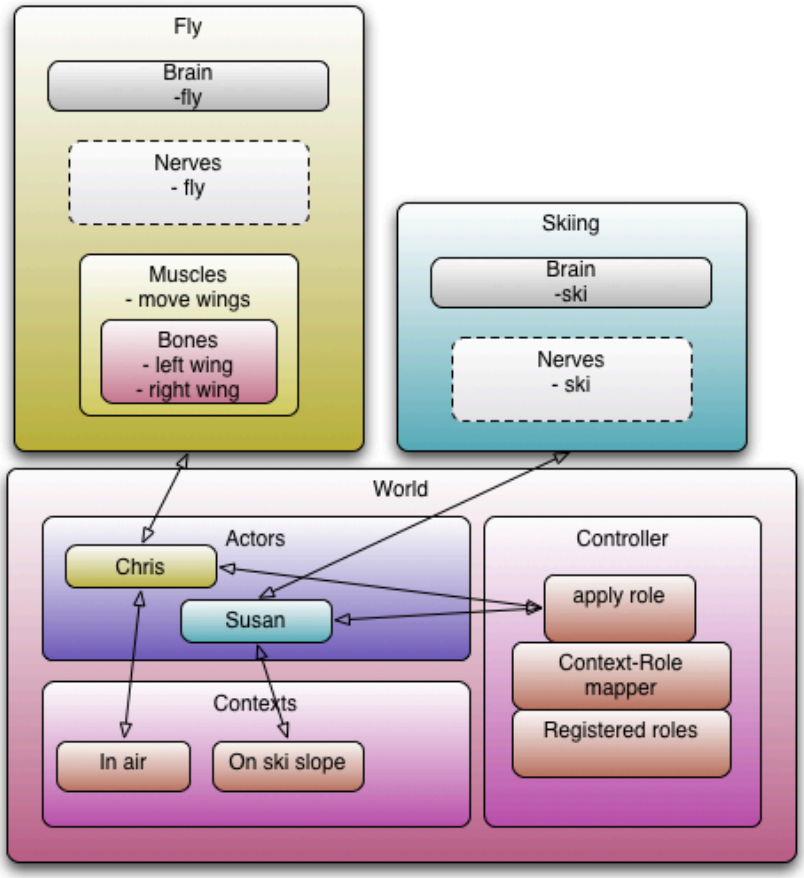

Figure 4 Organic context sensitive updates

Figure 4 shows how object operate as Actors within a World. Whenever an Object has a Context change, a World Observer, sends this event to the World Controller in order to figure out how to change the Roles of the object accordingly. The Context-Role mapper, takes the object, existing contexts, new context and existing roles and parameters and attempts to use this info to figure out which roles to apply on the object to suit the current situation.