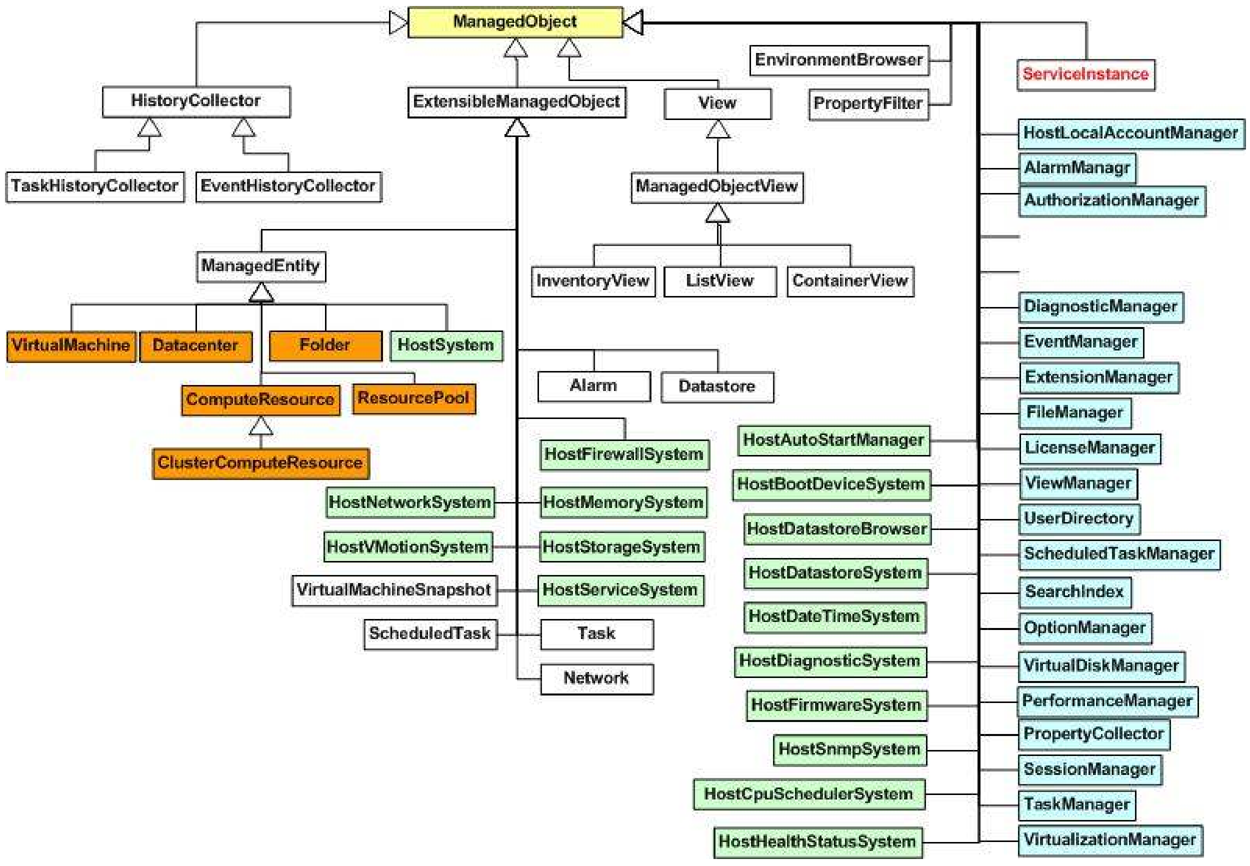
Object model of VI Java API

Steve Jin May 14, 2008

1. The overall object model



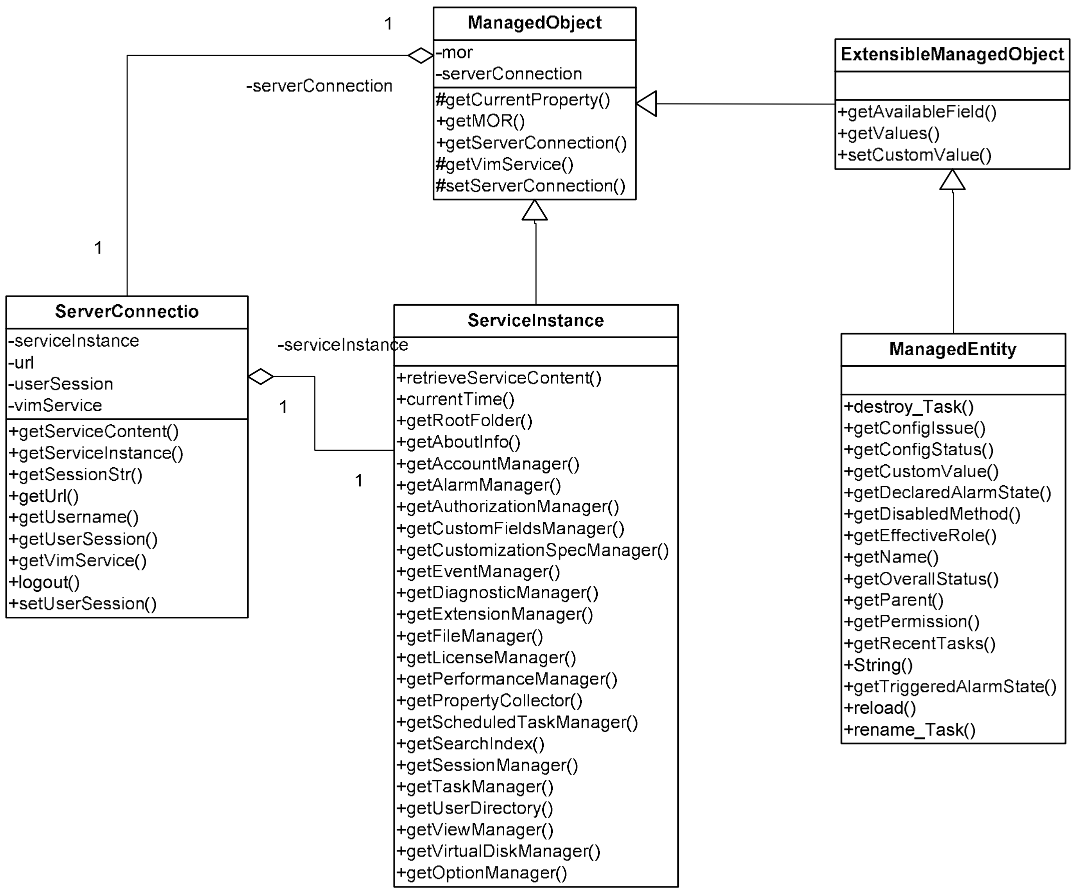
The above can be inferred from our VI SDK API reference. Please note that we don't have a managed object type called ManagedObject in our reference. This is a type defined to capture all the common properties and behaviors of all managed objects. Given the limited size, I only show the names of the types, not properties and methods.

To better group these managed object types, I used colors. On the right most side of the diagram are the ServiceInstance class and various "manager" classes like AuthorizationManager. From the ServiceInstance, you can get any object of these types with single call, for example getAuthorizationManager().

In the middle of left side, you can find ManagedEntity class and its sub-classes like HostSystem, VirtualMachine. These classes represent all the items you could find in the inventory tree from VI client. They are the most important managed objects in the whole model, and all tagged with orange color except the HostSystem.

The HostSystem is very much like ServiceInstance in that it has many "System" or "Manager" types closely attached to it, for example, HostDatastoreSystem. You can get hold of these objects with a single method call from a HostSystem object. For this reason, both HostSystem and all the attached classes are tagged the same color.

2. A detailed partial UML diagram



This UML diagram is extracted from the overall model but adds much more details with properties and methods. If you can understand this diagram, you can then easily understand all the other managed object types.

The ManagedObject class holds two private properties:

1. mor of type ManagedObjectReference -- pointing to the ManagedObjectReference object that is used to represent a managed object in VI SDK.

2. serverConnection of type ServerConnection -- pointing to the ServerConnection object I will cover later.

Besides accessors, the class has getCurrentProperty() method defined to encapsulate the PropertyCollector. This method gets called in subclasses to get a property. For example, the getName() in ManagedEntity called it like (String) getCurrentProperty("name"); In most of cases, you don't need to use it at all, I already provide explicit getter methods in concrete subclasses.

The ServerConnection is used to represent a connection to the server under a specific login user. It holds information like url to the server, the userSession with username etc., and vimService which is the JUMBO interfaces with 300+ methods. For convenience, ServerConnection also has a reference to a ServiceInstance object.

Now let us take a look at the ServiceInstance type. It's a special managed object and the first managed object you will have in a typical application logic flow. You can create a new ServiceInstance object by providing url/username/password, or url/sessionID combination. The later is not used as much as the first constructor, but very helpful when you develop a VI client plugin in Java. I will talk more about it in later blogs.

According to the API reference, the ServiceInstance has a ServiceContent object as its property, which holds all the ManagedObjectReferences to various "manager" attached to it, and an AboutInfo data object. Unlike web service interface, you can get any of them with a single call in Java API. ServiceContent object is, therefore, no long needed, and I don't even provide a getter to it.

At the right side of the diagram are the ExtensibleManagedObject and its subclass ManagedEnity. ExtensibleManagedObject doesn't have methods defined at all, but three properties. Therefore it only has three corresponding getters. As you will find in the overall UML diagram, it has many other subclasses, but I only list the ManagedEntity here.

ManagedEntity is one of the most important managed object type given that VirtualMachine, HostSystem etc. are all inherited from it. To search the inventory, I also provide a utility like class InventoryNavigator to group all the search methods to retrieve items in the inventory tree from a specified node. For example, you can easily find all the Virtual in one single call.

The above two ML diagrams should have given you a big picture about the object model and how key types are related to each other. If you really need to know more details, please download the latest jar file. It has all the binary, source code and some sample code.