P4FetchRequest : NSFetchRequest
Adds additional data to talk to the server. Additionally, a way to cancel the request and a block to execute when the request is finished.

P4ManagedObject : NSManagedObject
Adds property "endpoint" which indicates what command to run on the server for this entity

Category method:
to<ServerRepresentation> (for sending up to server)
localInstanceInContext: (calls the context version of the method)

overrides -
save: - This is to update the server
executeFetchRequest:error: - This is to retrieve items from the server
Category Method:
localInstanceOfObject: - copies MO from another store into this one

NSManagedObjectContext

NSManagedObjectContext

NSEntityDescription

Additional Data in the userInfo Dictionary for server operations.
"operations"
- fetch
- insert
- delete
- update
"route" the path in the server

P4ManagedObjectModel : NSManagedObjectModel

NSPersistentStoreCoordinator

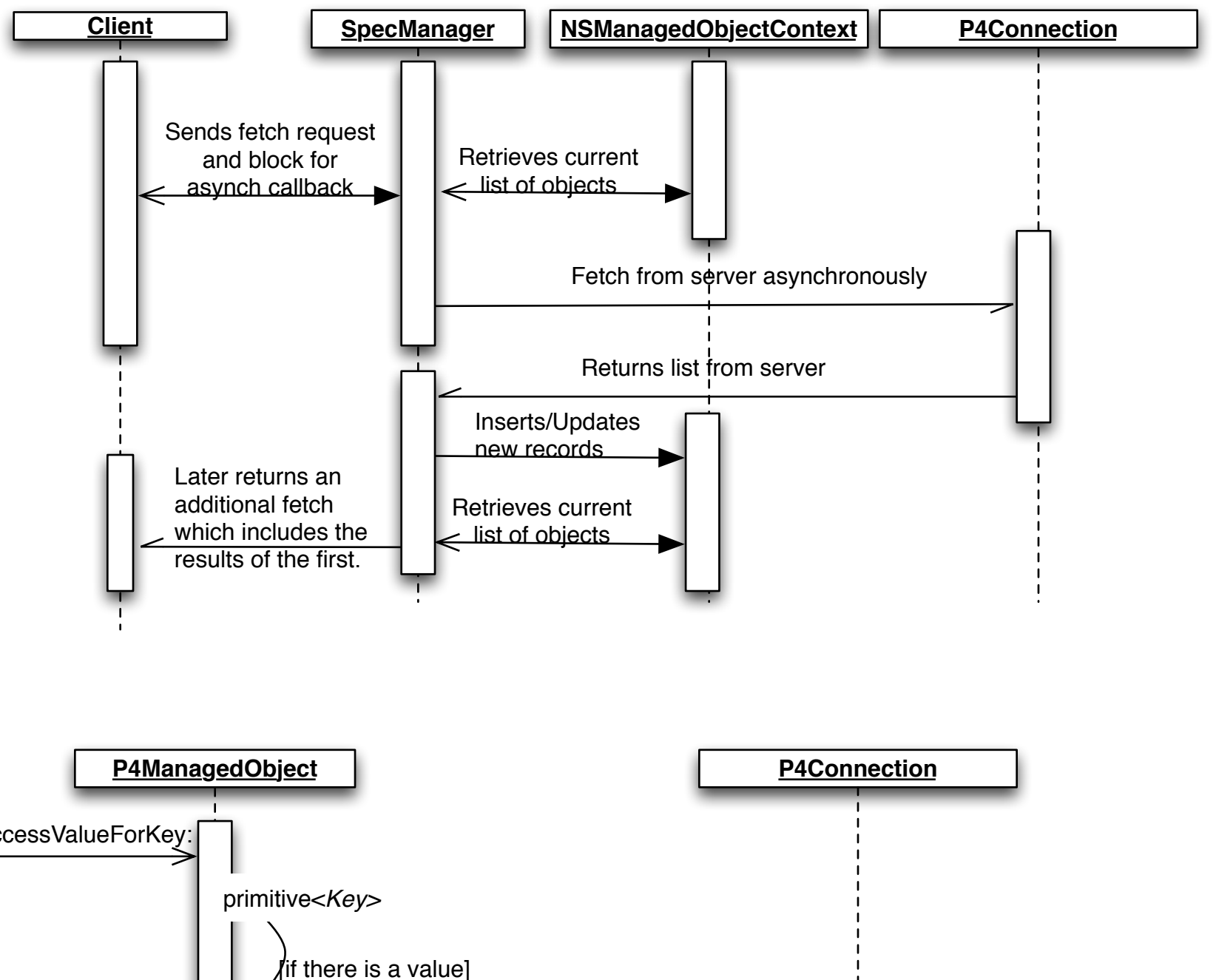
NSPersistentStore

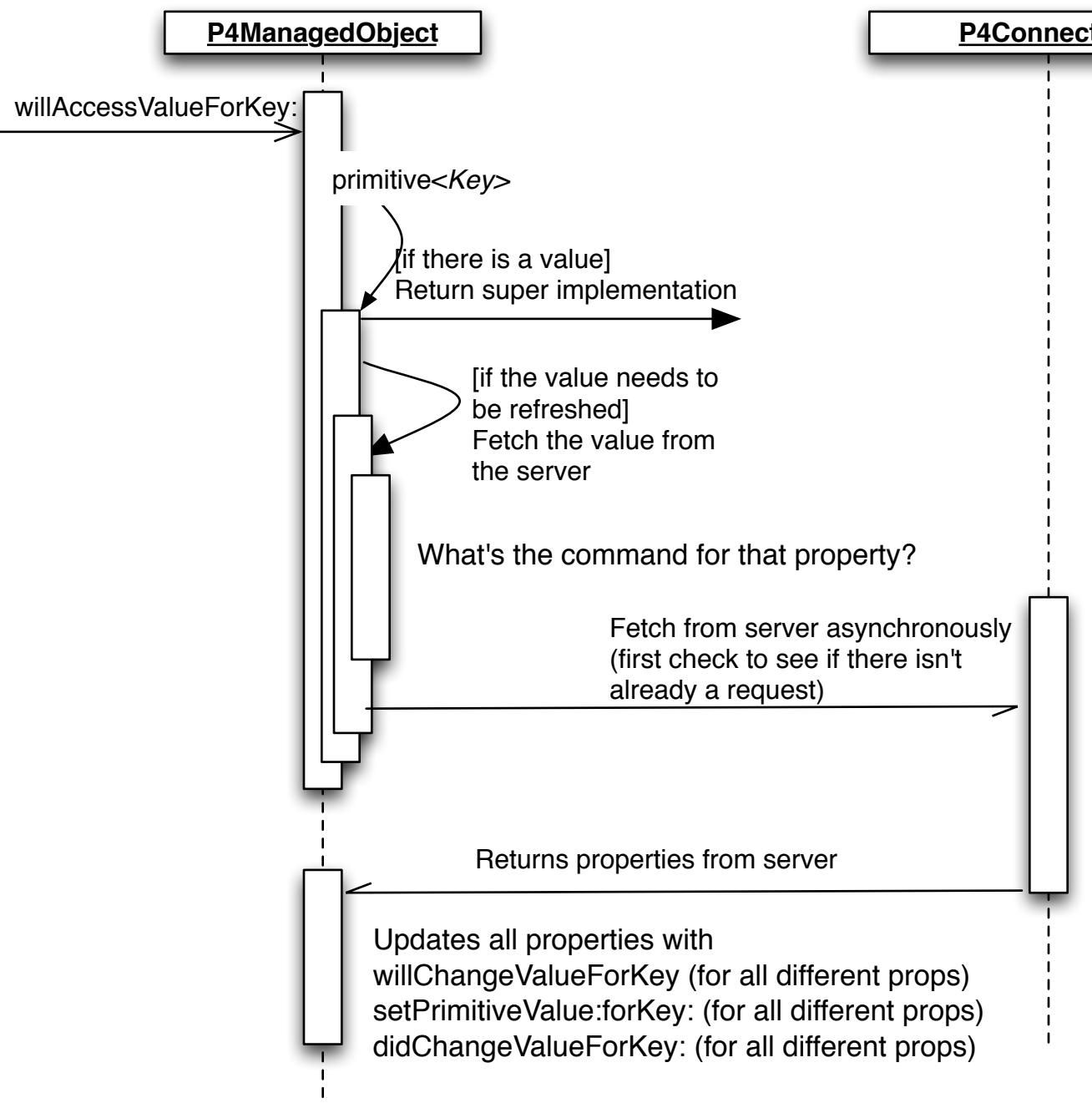
NSPersistentStore

Port

P4SpecManager
Generates an NSEntityDescription for a type
Provides translation between a specs field and the numeric fields
Provides the numeric fields for the id of the spec
Used to retrieve a spec
Uses a connection to talk to the server

The context first connects to the server and retrieves info and all the spec definitions. Then it uses the definitions to build an NSManagedObjectModel for all the specs.





In this example, we use the property name: "client".
Two keys per property.
- clientPeek
- client

When "clientPeek" is requested, it returns the primitive value of "client" and no server access occurs. We use primitiveKey:"client" because it will not invoke willAccessValueForKey: (which causes a server request). When "client" is requested, we return the value, but also start a request to the server to update the properties. When the request comes back, we emit key updates for "client" and "clientPeek".

In this way, the program can use KVO for the "client" property when it wants to trigger a server request and the "Peek" name to merely peek at the value without triggering a server request.

The non-peek versions of the property are what is saved to the store. The "peek" property names are transient.

When issuing a predicate to the model, you can specify the standard property names, but that might trigger a flurry of requests. You can also use a P4FetchRequest and include a server command-line and a block for completion. You might want to issue the fetch request with "peek" properties and the server command-line with real properties.

The implementation of the "peek" properties might be to update the Objective-C class at runtime but that might be difficult because we (might?) need to know the Class created for the entity. The other possible way is to use "valueForUndefinedKey:" to handle peek values.