1.1 Business Object Collection Pattern

1.1.1 Name and Source

Business Object Collection Pattern
Page 115-130 in the book "Web Service Patterns: Java Edition" [WSP 03]

1.1.2 Also Known As

-

1.1.3 Type

Micro-architectural design pattern.

1.1.4 Intent

To provide an interoperable web service that represents a collection of business objects that can be updated and queried in multiple ways, without exposing technical or language specific features to the clients.

1.1.5 Problem

A business collection containing business object can easily be updated in a typical object-oriented language such as java or c# by getting a reference to the object from the collection and then updating the object. You do not then also need to invoke an extra method call to update the object in the collection since the collection are referring to the same instance you were updating. In web services however, you do not get any remote references but all objects sent over SOAP from the server are instead reconstructed at the client as local instances with the data copied from the server. Further, when you are programming within your object-oriented language, there usually exist general and dynamic collection classes that can be used for adding and removing objects, but these generic classes may contain any kind of object and after retrieving these objects from the collection they will typically have to be downcasted when you want to invoke methods on them unless you just want to invoke a very general method that exists in the general baseclass such as “java.lang.Object”, if you would be programming in java. If you try to expose these kind of general collection classes to the clients there is a risk of problem e.g. if the clients are not using an object-oriented language and thus do not have the concept of classes and downcasting, but still are supposed to iterate such a collection and then downcast the retrieved objects when iterating the collection.

Often the number of potential objects in a collection is huge, and therefore you frequently want to provide some kind of filtering mechanism, i.e. you implement different query methods that can retrieve a subset of the business objects in a collection. If you are using an object-oriented language such as java or c# you can overload these query methods, i.e. the methods can have the same name but just differ by the number of parameters and their types. However, all kind of clients may not support method overloading, and WSDL 2.0 will not support overloading. To avoid overloading in the query methods you might then be tempted to only use one query method where the parameter is a string with a SQL (Structured Query Language) where clause, but that would not be very flexible since the clients would have to know low-level technical details of your implementation including database structure and you would not be able to easily change implementation without affecting the client applications, e.g. you might want to change the implementation from using a relational database to instead using an object database and use OQL (Object Query Language) instead of SQL as the query language.

When you are deleting or updating an existing object from the collection you also need a mechanism to tell the server which object you want to delete or update, considering that SOAP is a stateless protocol and you therefore can not rely on object references.

How can a web service represent a business object collection that can be updated and also queried in multiple ways while being interoperable and not exposing any technical or language specific features to the clients ?

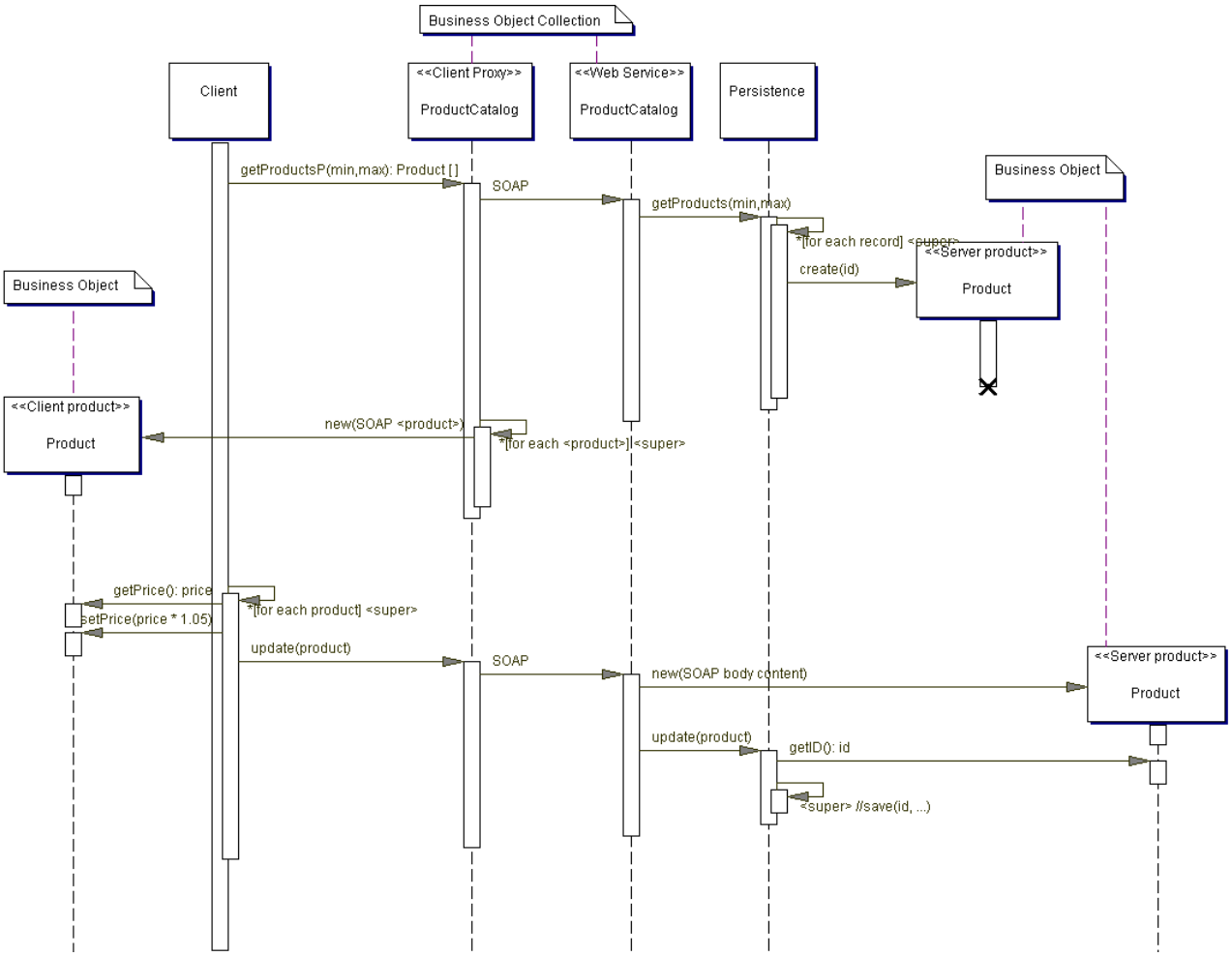
1.1.6 Forces

1.1.7 Solution

- If the client changes a business object in a business collection, the collection must also provide methods that let the client submit the updated data back to the server. The reason is that the objects will not be remote proxies but they will be copies of the data, as described in the “Business Object pattern”. Actually, a business object collection may be considered as a business object that contains other business objects as the non-primitive objects being discusses in the previous “Business Object pattern”. The necessary submitting back to the server is illustrated in the diagram below by the fact that it is not enough for the Client to invoke the method “clientProduct.setPrice” but it must also invoke the method “ProductCatalog.update” for the data to also be changed at the server.
- Type safe arrays should be used as parameters and return types instead of language specific collection classes or interfaces. For example, in the diagram below you can see that the “ProductCatalog.getProductsP” method return an array “Product[]” instead of something like “java.util.List”.
- Methods for querying collections are often overloaded within implementations, but you should not overload web service methods, i.e. do not expose methods that only differ by parameter types. WSDL 1.1 supports operation overloading but WSDL 2.0 will not, according to the “WSDL 2.0 Working Draft 2004-03-26”. Another reason that the method name should also differ is that all programming languages do not support overloading. For example, in the diagram below you may use method overloading within your implementation and have many methods named “getProducts” in the “Persistence” class, but in the Web Service interface the methods must have different names, for example a method “getProductsP” may return a collection with all products within a certain price interval, while a method “getProductsN” may return a collection with products that have a name containing some substring provided as parameter.
- In the interfaces for querying a collection, do not expose any technical details to the clients but instead use general concepts in the interface that might be implemented in different ways. For example, if you would let the web service clients use a method call such as “getProductsByUsingAnSQLwhereClause(‘prodtable.price >= 100 and prodtable.price <= 200’)” then you would get problems if you e.g. are implementing in java and would want to change your implementation from JDBC to JDO. An implementation with SQL code in the client would also violate the “Layers” pattern. [POSA 96]. As indicated in the sequence diagram below, the clients should instead be able to make a method call like “getProductsP(100, 200)” where the parameters are the min and max value for the price.
- Each business object within the collection should have a unique key (“Identity Field” [PEAA 02]) that can be used for identifying the object when it is going to be updated or deleted. The usage of the product key is illustrated in the diagram below within the method “Persistence.update” and the value of the id key may be used in an sql where clause statement in the “save” method. The value of the id will typically (but not

necessarily) be the primary key for an entity in a relational database. In a query the id will be returned within the SOAP response body, and then reconstructed at the client by the proxy class. When the client later submits the entity back to the server the same id will be contained in the SOAP request message, and then reconstructed again at the server when the SOAP body is parsed. As illustrated in the diagram below, the product objects at the server will somehow have to be recreated in the SOAP message since SOAP is a stateless protocol, although not necessarily by invoking constructor calls as in the diagram but the products could instead be picked up from an object pool with product instances.

The sequence diagram below illustrates a use case where the price of a product is increased with 5% for each product within a certain price interval. The products are business objects and the product catalog that contains the products is a business object collection. Note that the solution in the example should not be used in the real world because it is bad from a performance point of view, since you should not do many RPC (remote procedure calls) like SOAP invocations in a loop, but rather let the data to be submitted in a value object according to the “Data Transfer Object pattern”. The purpose of the RPC in the iteration is just to illustrate the discussed concepts above, such as using type safe arrays and submitting changes back to the server.



[the occurrence of "<super>" in the diagram is not intentional and will be removed in the final version]

Client – A client class that uses the business object collection proxy class for getting and updating business objects within the collection by submitting the local object changes back to the server.

ProductCatalog<<Client Proxy>> – A client proxy object that locally represents the business object collection exposed as a web service. It communicates with the ProductCatalog<<Web Service>> by sending and receiving SOAP messages.

ProductCatalog<<Web Service>> – A business object collection exposed as a web service. It communicates with the ProductCatalog<< Client Proxy>> by sending and receiving SOAP messages. Typically the collection will not actually contain all the business objects in the collection but instead provide query methods that can retrieve a subset of these objects.

Persistence – A server sided class that is responsible for the technical implementation that retrieves and saves objects to and from some persistent storage. Typically this class uses SQL for mapping business object to a row in a relational database. Instead of letting the ProductCatalog (business object collection) be dependent on one such persistence class, you may use an interface instead and let a factory class create an object that implements the interface, for example an RDB implementation. This is being described in the "Data Access Object pattern" [CJP 03].

Product<<Server>> – A business object in the business object collection (ProductCatalog). The objects typically only live within an object invocation, even though they also might exist within an object pool. This is because of the fact that SOAP messages are stateless in nature and does not support remote references but the data is only copied in the SOAP XML messages.

Product<<Client>> – A business object that is a local copy of the corresponding business object at the server. Changes to this local object does not automatically update the server sided business object but it must be submitted back to the server. The class ProductCatalog<<Client Proxy>> can convert this object to an XML structure sent with SOAP to the server.

1.1.8 Consequences

Different clients can retrieve copies of business objects from the collection but the pattern does not provide any mechanism for notifying other clients when their current copy of the data has been changed by some other client.

Since a business object collection can contain very many business objects you can get significant performance problems if you do not provide query methods that retrieves subsets of the collection. Therefore junior programmers should be informed of this potential problem to make sure that they will not ignore or forget it until the problem shows up in a production release when the number of objects can be much bigger than during tests.

1.1.9 Related patterns

This pattern is similar to the "Business Object pattern" since a business object collection may be considered as a business object that contains other business objects as the non-primitive objects being discusses in the "Business Object pattern".

Usually it is better to expose more coarse-grained objects and methods representing business processes, e.g. to use the "Business Process (Composition) pattern", instead of exposing a business object collection as a web service.