A MORE RATIONAL WAY TO PROGRAM STORAGE AND COMMUNICATION OF OBJECTS.

By Ir J.A.J. van Leunen

Retired physicist & software researcher

Location: Asten, the Netherlands

Website: www.scitech.nl

Communicate your comments to info at that site.

ABSTRACT

A C# class library is described that offers an efficient and secure way of object oriented data transfer and data storage. The classes convert a relational database in an effective object oriented database and a file system in an object oriented data storage and transfer system.

THE CLASS LIBRARY

Software uses many kinds of data. Sometimes it concerns worksheet data and sometimes it is data that is retrieved or generated in some way or it are results of actions. Often these data are interrelated. In that case the ensemble forms a complex object. When this is the fact then it becomes worthwhile to attach the actions that are regularly applied to the data to the object that contains the data. With comparable data structures the same will happen. The object oriented software generation uses this approach intensively.

In software programs data is often exchanged between locations or they are stored in documents, files or databases and retrieved at instance or received at another location. When a complex object is stored or communicated it must first be unravelled in simple data types that can be handled by the storage and communication services. These services can only handle simple data types. Apart from that usually not all the data that is contained in the object needs to be treated. For example private data such as worksheet service data must not be sent or stored. Only public data should be handled. So the programs must handle the unravelling of the objects. At retrieval or receipt of the data the object must be restored. Often the objects have a hierarchical structure consisting of items that just contain simple data and items that contain collections of simple data and/or complex objects. An automobile has such a complex structure. Its specification will reflect this. On top of all of this, the way that the programming language treats storage in files, storage in databases, storage in documents and communication over networks in quite different ways. This means that a large part of the program may be involved with unravelling and restoring the objects.

This is odd. It is better to let the objects do this job. Instead of implementing this capability for every type of object it is sensible to derive such objects from a specialized small set of super classes of objects that do the job. The objects in this super class must be treated in a special way. The programming languages do not normally provide this. The treatise is different for storage in files, storage in databases, storage in documents and communication over networks. When the object is restored the non-treated data must get a sensible initial value. In order to solve this, the data fields of the derived objects will be enriched with parameterized attributes that indicate to the routines of the super class object whether and how the data must be treated. Both java and the C# programming language support the attachment of parameterized attributes to the data fields of objects.

This approach will not only rationalize the storage and transfer of complex data. It will also increase the robustness of the programs. It will reduce programming time with a significant factor. It is relative easy to let an object class that is derived from such a super class specify its own database structure. When the structure of the object is altered its class can adapt the corresponding database structure. The encapsulating object can store itself and all its elements in the database and it can remove itself and its elements from the database. It can retrieve itself from the database and restore itself including the data that was not stored. This solution converts a relational database in a hierarchical database and, when the object classes are used, in an effective object oriented database.

The approach uses the public fields of the objects that are used in an object oriented language such as C# or java. When a field must be private then it can be protected by a corresponding property that offers get and/or set routines. Properties can also be used to store multiple bits in an integer that represent Booleans. Such a Boolean can indicate whether something is present or not. This is used to indicate for example that the object is a pure collection. Other Booleans indicate then whether the collection contains sorting keys that can be used for indication or enumeration. In order to enable hierarchical storage and retrieval in relational databases, every object of the super classes contains two ID's. One ID is a GUID and represents the object. The other ID is a different GUID or null and represents the parent of the object. A pure collection is not stored itself in the database. Only its contents is stored. On retrieval the collection object retrieves the content. An object must only be stored once in the database. For that reason a special super class of reference objects exists whose objects can be stored instead of the duplicates.

The encapsulating object is special. It can do special tricks. It can convert its contents in a binary stream, compress the stream and encrypt the result after which the stream is converted into a base64 string. The compression is always done when encryption is performed and never done when the stream has a length that does not surpass 256 bits. The encryption is optional. The conversion into a base64 string is only done when the stream is transferred. At receipt or retrieval the stream is decoded. The encryption key must be known at both sides of the channel. Normally the encapsulating objects only appear at the top of the hierarchy. However they are also normal complex objects and may appear lower in the hierarchy as well. The wheel of an automobile may also be taken as an encapsulating object. Under control of the parent object the internal objects can convert their contents in a binary stream or store its contents in a database. They can restore themselves from the binary stream or retrieve themselves from the database. They can also delete themselves from the database. In each case the parent object controls this activity. The internal objects contain an optional placeholder for a string that acts as an identifier and an optional placeholder for an enumerator. The container objects use these data for their sorting keys. These keys are also optional. When they are not in use the placeholders for key elements and key arrays are set to nil pointers.

Storage and retrieval of documents is done using XML technology. The encapsulating object can create its XML file. Its class can create the corresponding XSD file. In principle it is also possible to create one or more XSL files.

The implementation consists of four super classes. All classes including the derived application classes inherit from the complex object super class that represents normal internal complex objects. A second super class represents pure containers. Also this class derives directly from the first super class. The third super class represents references to complex objects. It derives directly from the first super class. The fourth super class represents encapsulating complex objects. Like the other super classes it derives from the first super class. Further the implementation contains a small library of routines and programs. The routines are used by the classes and can be used by application programs. One of the programs helps starting and removing a database.

It is obvious that the implementation costs extra overhead. In large applications this overhead is far compensated by the reduction of the complexity, the decrease of programming effort, the increase of the robustness and the ease of extension of the produced programs.

CODE

The library is available for free access and usage.

An older version that is well proven is stored in:

https://docs.google.com/leaf?id=0B8ZNOnNHFrbrNzRlMDgwMGEtZmE2NS00ODAwLTg3OWItOTliNTc3NmY5NTAz&hl=en .

A new version that uses generics, but that is not intensively tested is stored at:

https://docs.google.com/leaf?id=0B8ZNOnNHFrbrNzRlMDgwMGEtZmE2NS00ODAwLTg3OWItOTliNTc3NmY5NTAz&hl=en .