



Open CASCADE Technology
6.7.1

Distribution of Data Through OCAF Tree

April 29, 2014

Contents

1	Introduction	1
2	Description	2
3	Comparison and analysis of approaches	3
4	Conclusion	5

1 Introduction

Open CASCADE Application Framework (OCAF) represents a set of classes for Rapid Application Development. OCAF helps to implement such functions as undo and redo, copy, cut and paste, storage and retrieval of documents, and many others.

OCAF is based on a notion of label-attribute. The labels form a tree. The attributes are attached to the labels and store user data in the OCAF document.

This document describes how data should be stored in the OCAF document and considers the following issues:

- Should we choose standard attributes or is it better to create new ones?
- How can we optimize data allocation to make it easy to use and efficient for memory usage and application speed?

In particular, the document describes an example of using standard OCAF attributes and creating new OCAF attributes. It is assumed that the reader is already familiar with some OCAF basics.

2 Description

When you start to design an application based on OCAF, usually you need to choose between standard and newly-created attributes to be used for allocation of data in the OCAF document. Before giving an answer to this question, let's overview standard OCAF attributes.

All basic data types are represented in OCAF as special attributes:

- Integer,
- Double,
- String,
- Array of integer values,
- Array of double values,
- Array of string values,
- Topological shapes.

An attentive reader might have noticed that there is no Boolean type. This is because the Integer type is usually used instead.

So, we have an opportunity to describe any model by means of standard OCAF attributes. But will this description be efficient for memory, speed, and, at the same time, convenient to use? It depends on a particular model.

OCAF has one restriction: only one attribute type may be allocated to one label. It is necessary to take into account the design of the application data tree. For example, if a label should possess several double values, it is necessary to distribute them through several child sub-labels or use an array of double values.

Let's consider several boundary implementations of the same model in OCAF tree and analyse the advantages and disadvantages of each approach.

3 Comparison and analysis of approaches

Below are described two different model implementations: one is based on standard OCAF attributes and the other is based on the creation of a new attribute possessing all data of the model.

A load is distributed through the shape. The measurements are taken at particular points defined by (x, y and z) co-ordinates. The load is represented as a projection onto X, Y and Z axes of the local co-ordinate system at each point of measurement. A matrix of transformation is needed to convert the local co-ordinate system to the global one, but this is optional.

So, we have 15 double values at each point of measurement. If the number of such points is 100 000, for example, it means that we have to store 1 500 000 double values in the OCAF document.

The first approach consists in using standard OCAF attributes. Besides, there are several variants of how the standard attributes may be used:

- Allocation of all 1 500 000 double values as one array of double values attached to one label;
- Allocation of values of one measure of load (15 values) as one array of double values and attachment of one point of measure to one label;
- Allocation of each point of measure as an array of 3 double values attached to one label, the projection of load onto the local co-ordinate system axes as another array of 3 double values attached to a sub-label, and the matrix of projection (9 values) as the third array also attached to a sub-label.

Certainly, other variants are also possible.

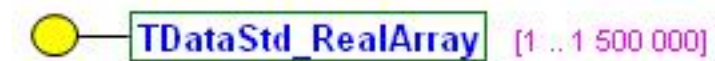


Figure 1: Allocation of all data as one array of double values

The first approach to allocation of all data represented as one array of double values saves initial memory and is easy to implement. But access to the data is difficult because the values are stored in a flat array. It will be necessary to implement a class with several methods giving access to particular fields like the measurement points, loads and so on.

If the values may be edited in the application, it means that the whole array will be backed-up on each edition. The memory usage will increase very fast! So, this approach may be considered only in case of non-editable data.

Let's consider the allocation of data of each measurement point per label (the second case). In this case we create 100 000 labels – one label for each measurement point and attach an array of double values to these labels:



Figure 2: Allocation of data of each measurement point as arrays of double values

Now edition of data is safer as far as memory usage is concerned. Change of value for one measurement point (any value: point co-ordinates, load, and so on) backs-up only one small array of double values. But this structure (tree) requires more memory space (additional labels and attributes).

Besides, access to the values is still difficult and it is necessary to have a class with methods of access to the array fields.

The third case of allocation of data through OCAF tree is represented below:

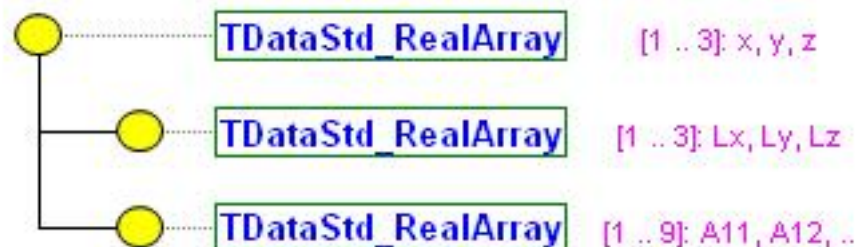


Figure 3: Allocation of data into separate arrays of double values

In this case sub-labels are involved and we can easily access the values of each measurement point, load or matrix. We don't need an interface class with methods of access to the data (if it exists, it would help to use the data structure, but this is optional).

On the one hand, this approach requires more memory for allocation of the attributes (arrays of double values). On the other hand, it saves memory during the edition of data by backing-up only the small array containing the modified data. So, if the data is fully modifiable, this approach is more preferable.

Before making a conclusion, let's consider the same model implemented through a newly created OCAF attribute.

For example, we might allocate all data belonging to one measurement point as one OCAF attribute. In this case we implement the third variant of using the standard attributes (see picture 3), but we use less memory (because we use only one attribute instead of three):



Figure 4: Allocation of data into newly created OCAF attribute

The second variant of using standard OCAF attributes still has drawbacks: when data is edited, OCAF backs-up all values of the measurement point.

Let's imagine that we have some non-editable data. It would be better for us to allocate this data separately from editable data. Back-up will not affect non-editable data and memory will not increase so much during data edition.

4 Conclusion

When deciding which variant of data model implementation to choose, it is necessary to take into account the application response time, memory allocation and memory usage in transactions.

Most of the models may be implemented using only standard OCAF attributes. Some other models need special treatment and require implementation of new OCAF attributes.