Experience with Object Oriented methodologies in the new Online Event Display for Delphi

Serge Dù and Christian Arnault
LAL - Orsay, France

An Object Oriented modelling methodology has been widely used in the building of the new Delphi Online Event Display, both in the early phase of the software life-cycle (ie. requirements specification and design) and in the implementation of the application's internal architecture (ie. from the memory management up to the graphical user interface).

This method is supported by a tool which generates C++ code and allows to incorporate existing software such as Zebra based components. It was then possible to benefit from the experience of detectors' experts, starting with a limited number of detectors, and to concentrate on the design of a general and extensible framework.

Class-Relationship model:

The Class-Relation model [1], (supplied by Softeam, a french company) is an extension of the Entity-Relationship model that uses the concept of inheritance found in the Object model.

The derived Object Oriented methodology promotes the usage of that model since the early stage of software development, namely the requirements specification phase. It provides a formalism that is used from that stage up to the detailed conception and implementation steps.

Figure 1: Schema Detectors – tree structures – detailed design.

A Schema is a set of Classes which belong to the same conceptual domain. It is a way to structure complex applications in modules easily manageable. That example shows a Class
Device composed of 0-to-many Devices, seen as subdetectors. It is the Class-Relationship model of a tree structure. EventHeader and RootDevice are the roots of the trees. They inherit from Device and DataBlock.

Objecteering:

This method is supported by a tool named Objecteering [2] which consists of:

- a graphical editor for the semantic of the whole application, in terms of classes and relationships
- a syntactical translator to express the formalism in a source code file.
- filters to have a customized view of the code: user of a class, heir of a class, at the Class-Relation syntactic level or at the C++ code level.

![Figure 2: Graphical editor and syntax example](image)

The code generator lets the implementation part of the methods to the programmer, expected to use C++ code. It is the place where existing software components can be incorporated. In the Delphi project, we used that possibility to hook up two kinds of existing software:

- user written functions, for data access and decoding facilities, or geometry definition functions.
the Delphi OED project

For the new Online Event Display in Delphi, it was decided to use the Class-Relation method and translator at least for the specification phase and early development steps, until a prototype could be achieved.

A first release, with a very limited number of detectors and functionalities, demonstrated that the prototype was in fact a partial but realistic implementation of the complete application's specifications and modelling.

Therefore, the decision has been taken to go on with this methodology for the next development step.

![Diagram of the Delphi OED project](image)

Figure 3: Schema Views - external view - analysis

Conclusions

After a short-medium period of development, we are now in position to identify the benefits that can be expected in the use of that Object Oriented methodology:

- inheritance and virtual methods are essential to implement and test specifications' parts when related classes are just outlined.

- the definition of generic classes (like 'Detector' or 'Volume') allows to implement in a straightforward way new sub-detectors, in a consistent way with the overall development.

- the identification of schemas (group of classes) allows to define clearly priorities between different specification's implementations. It provides then some help to setup a development strategy. The reliability of the methodology on long-term
period has still to be tested and proven; to see how it can help to upgrade the
application in order to satisfy requirements’ evolution, because that is become a
corner stone in software development.

Acknowledgments

We would like to thank P Gavillet, JP Laugier and P Charpentier, from the DAS
group in Delphi for their precious help.

References


Wesley 1990.
