GG

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CERN LIBRARIES, GENEVA

CERN/ECP 94-15
10 October 1994

CERN-ECP-94-15

A CONFIGURABLE CODE GENERATOR FOR OO METHODOLOGIES

A.Aimar, A.Khodabandeh, P.Palazzi, B.Rousseau

CERN ECP/PT

Abstract

Automatic code generation is a way to ensure consistency between design and implementation stages of the software life cycle, but most CASE tools offer weak code generation capabilities. In the OPLA project, we have built a prototype of a configurable code generation system that can be coupled with most CASE tools and which can be parametrised to generate any language or format. The system relies upon a central dictionary that preserves independence between input and output formats. Versions have been produced for OMTool and LOV/OMT and we generate C++, ADAMO DDL, and World Wide Web documentation.

Presented at *Computing In High Energy Physics*, San Francisco, April 21-27, 1994.

A Configurable Code Generator for OO Methodologies

A.Aimar, A.Khodabandeh, P.Palazzi, B.Rousseau
rousseau@ptsun00.cern.ch
Programming Techniques Group
CERN, ECP Division, 1211 Geneva 23, Switzerland

Automatic code generation is a way to ensure consistency between design and implementation stages of the software life cycle, but most CASE tools offer weak code generation capabilities. In the OPLA project, we have built a prototype of a configurable code generation system that can be coupled with most CASE tools and which can be parametrised to generate any language or format. The system relies upon a central dictionary that preserves independence between input and output formats. Versions have been produced for OMTool and LOV/OMT and we generate C++, ADAMO DDL, and World Wide Web documentation.

OO METHODOLOGIES AND LANGUAGES

Whilst Object-Oriented technology becomes more and more popular, it suffers from a crucial gap between the methodologies and languages because most OO languages do not offer native support for high level abstractions and notations.

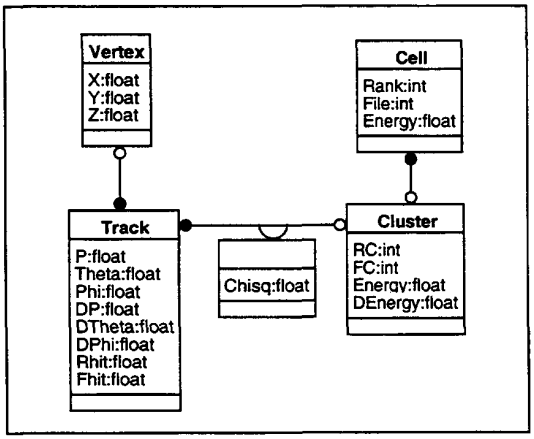

Figure 1. An OMT Object Model, produced with OMTool. Most CASE tools have good graphical editors to draw such diagrams.

Considering the Object Modeling Technique (OMT [1]) as an example, concepts such as association or aggregation have no equivalent in C++ or Eiffel. Even the generalisation/specialisation abstraction cannot be mapped directly onto an OO language because it contains more semantics than inheritance. Current OO methodologies are fuzzy about the way such abstractions should be implemented with most choices being left to the programmer.

To overcome this problem CASE tools usually provide code generators, although in most cases this facility is restricted to the production of simple classes and method templates. This is not satisfactory for three reasons:

- The mapping between the classes of the analysis (Figure 1.) and the classes of the language (Figure 2.) is too simple and often not safe.
- No generic run time support is generated. For instance, associations between classes are commonly mapped to pointers, but the operators to manipulate the associations are left to the programmer (e.g. add or remove an association, check cardinalities, navigate along an association). General purpose methods to display instances, to manage subsets of instances of a given class, or to print comprehensive error messages must be programmed from scratch.

- Code generators are usually not configurable and they only target a restricted set of languages, commonly C++ and SmallTalk.

```
class Cell
//  Calorimeter Cells
{
protected:
int Rank; // Rank or Row
int File; // File or Column")
float Energy; // deposited in cell
Cluster* ptrCluster;
};
```

Figure 2. A C++ class generated by OMTool [2] out of the Object model of Figure 1. The association with the class "Cluster" is mapped to a simple pointer, but no methods are generated to handle the association, or to check attributes ranges.

FILLING THE GAP: THE OPLA PROJECT

In the OPLA project (Object Programming LAboratory) of the CERN Programming Techniques Group, we are experimenting with possible components of a seamless OO development environment. Our primary objective is to let the programmers think and implement in terms of the high level abstractions provided by the OO methodologies. To achieve this goal, three main ingredients are required:

Well-designed mappings between various OO methodologies and languages.

We are implementing such a mapping for the Object Model of the OMT methodology and the C++ language, which are both becoming de facto OO standards. The mapping provides a comprehensive set of operators to manipulate entities of the OO Methodology.

Using these operators, the programmer can focus on the high level objects and is not concerned by too many implementation details. For instance, the OPLA mapping provides operators for adding and removing association links between instances and to navigate along those links. The mapping avoids the pitfalls of C++ (e.g. pointers), and allows the compiler to perform full static checking.

It shares many features in common with ODMG, the Object Database Standard [3].

An Open Architecture.

The OO technology is still evolving so we need a suitable framework for experimenting with new ideas and products and to try them out on particle physics software. The OPLA architecture is able to integrate commercial tools (CASE tools, compilers, development environments) and to couple them with the OPLA tools.

Commercial CASE tools (such as OMTool), are used to draw the models of the application, and to generate the corresponding data dictionary. The OPLA C++ code generator then translates this dictionary into C++ code. This includes:

- the declaration of the classes of the application,
- companion classes to handle sets,
- methods to, for instance, access attributes and navigate associations.

Programmers may use C++ development environments of their choice to produce the C++ methods of the application. This code uses classes and calls methods that have been automatically generated. It also calls generic methods defined in the OPLA Library (OPLib). A C++ example is shown in Figure 5.

Automatic Code Generation.

If it integrates a well-designed mapping between the methodology and the target language, automatic code generation enforces the methodology, guarantees the consistency within the design-to-coding loop, and lets the programmer concentrate and work with abstractions. We have built a prototype of a configurable code generation system that can be coupled with all CASE tools that provide a textual dictionary format or a programing

interface to access their dictionary (most tools do) and which can be configured to generate any language or format.

THE OPLA CONFIGURABLE CODE GENERATOR

The code generator works in a quickly evolving environment: new CASE tools are appearing on the market. They support various OO Analysis and design methodologies that are themselves evolving. The code generator thus has to be coupled with various CASE tools. In addition, OPLA will consider several target languages and, for each of them, several mappings to the OO methodology.

Because we want to avoid writing a new code generator for each new CASE tool and each new target format, the code generator has to accommodate these changes easily, and thus should be configurable.

Independence between input and output formats

In order to facilitate the integration of new CASE tools, input dictionary formats and output language formats must be independent. For instance, the C++ generator can work without any knowledge of the CASE tool used to produce the dictionary.

This is achieved through an architecture based upon the OPLA Dictionary Format (ODF) that has an OO methodology dependent, but CASE tool independent structure. Plugging in a new CASE tool only requires writing a front-end to convert its dictionary format to ODF. For instance, a front-end converts OMTool dictionaries, and another one converts LOV/OMT dictionaries [4]. They both produce an OPLA dictionary.

The OPLA code generator provides an ODF parser that fills in a database and an interpreter that lets you access the database and perform formatted output. Thus, generating code for a new language only implies writing a script that navigates the database and generates appropriate text (see example in Figure 4.). For instance there is a generator for producing C++ and another one to produce ADAMO DDL[6], etc.

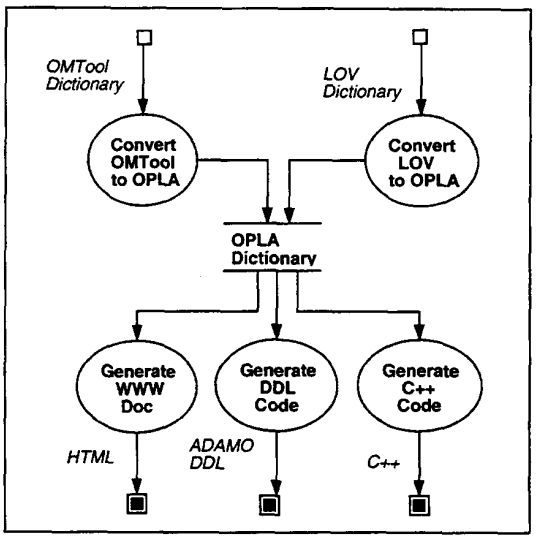

Figure 3. The code generator is made of a set of converters and generators. A converter transforms a particular dictionary format to the OPLA dictionary format. A generator navigates an OPLA dictionary and produces formatted output.

```
(foreach class cl
  (openfile out "~A.h" [cl :name])
  (format out "class ~A~%" [cl :name])
  (format out "// ~A~%" [cl :name])
  (format out "{~%")
  (format out "protected:~%")
  (foreach attribute att
    in [cl :attributes]
      (format out "  ~A ~A; // ~A~%"
        [att :type] [att :name]
        [att :comment]))
  (foreach role ro in [cl :roles]
    (format out "  ~A* ptr~A~%"
      [ro :class] [ro :name]))
    (format out "};~%")
    (closefile out))
)
```

Figure 4. This simple code generation script produces simple C++ class headers, as the one in Figure 2.

Tolerance to changes in the model

Methodologies and CASE tools are evolving;

their data models are continuously improving and growing. The code generator is able to accommodate these changes with minimal programming effort.

Changes to the Object Model are handled by reflexivity: the ODF parser used by the generators is itself automatically generated out of the OPLA Meta Object Model (the object model that represents classes, attributes, associations and so on in terms of classes). Any change to this Meta Model is propagated automatically to the code generator.

Current status

The code generator is written in Lisp. We have implemented front-ends for OMTool and the LOV/OMT. Configurations exist for the OPLA C++ mapping, ADAMO Data Definition Language, and WWW. documentation.

```
#include "Cluster.h"
#include "Cell.h"

void PrintCells (ClusterSet &
SetToDisplay)
{
 Cluster clu;

 FOREACH ( clu, IN, SetToDisplay,
 {
  cout << "Printing Cells of Cluster: "
       << clu.ID();
    clu.Cells().Display();
 } )
};
```

Figure 5. Example of an OPLA C++ function written by the user. For each Cluster of a given set of Clusters, this function displays the associated calorimeter Cells. Include files are automatically generated. The FOREACH macro allows loops over sets of objects. The method "Cells" of the class "Cluster" returns the set of all cells associated with an instance of "Cluster". This method is generated automatically. The method "Display" of the class "Set" is inherited by the class "ClusterSet". It is a generic method that works for all kind of sets, and is defined into OPLib.

Further work is needed to include new CASE tools (e.g. StP) and new languages (SmallTalk, Common Lisp Object System) in our architecture.

Another improvement issue concerns the storage of diagram layout information into the OPLA dictionary format. This will allow the generation of dictionaries for other CASE tools.

CONCLUSION

As part of the OPLA experiments, we have built an advanced prototype of a configurable code generator that can be configured to accept various CASE tools dictionary formats and to generate a wide range of languages or formats. In particular, the code generator produces C++ code out of OMT Object models according to the OPLA C++ mapping. A first prototype of the mapping is being tested.

Experience shows that configuring the code generator for a new target format is a quick and easy task, although plugging in a new CASE tool is a more complex task that requires the writing of a parser.

The code generator facilitates to a large extent the definition of the mapping as it is very easy to configure in order to experiment with different solutions and variations. It also encourages the development of mappings with other languages.

REFERENCES

[1] Object-Oriented Modeling and Design, J.Rumbaugh et al. ISBN 0-13-629841-9, Prentice Hall, Englewood Cliffs, NJ 07632 USA

[2] OMTool, Advanced Concepts Center, 640 Freedom Business Center, PO Box 1561,King Of Prussia, PA 19406, USA.

[3] The Object Database Standard: ODMG-93, ISBN 1-55860-302-6 Morgan Kaufmann Publishers, San Mateo, CA 94403, USA

[4] LOV/OMT, Verilog SA, HQ: 150, Rue N. Vauquelin, BP 1310, F-31106 Toulouse cedex.

[5] Relations as Semantic Constructs in an Object-Oriented Language, J.Rumbaugh. OOPSLA'87 Proceedings, p. 466.

[6] ADAMO, ER Programming System, URL address: http://www1.cern.ch/Adamo/