U.P.B. Sci. Bull., Series D, Vol. 70, No. 3,2008 ISSN 1454-2358

INFORMATIC SYSTEM FOR NUMERICAL COMMAND PROGRAMS AUTOMATIC GENERATION

Corneliu NEAGU[1], Victorin CONDEI[2]

Existența unei mari diversități de produse CAD/CAM comerciale, având caracteristici de utilizare echivalente, generează dificultăți pentru firme în alegerea celui mai potrivit produs pentru nevoile sale concrete. Pe de altă parte, produsele software oferite de producătorii consacrați necesită investiții substanțiale de achiziție şi de formare a personalului desemnat să le utilizeze. De aceea, în cazul întreprinderilor mici şi mijlocii se recomandă dezvoltarea personalizată a unor soluții de generare automată a programelor 'NC', utilizând librării grafice gratuite de tipul 'Open' (OpenGL, OpenCascade) şi un limbaj de programare adecvat aplicațiilor grafice. În baza acestui obiectiv major, în lucrarea de față se propune un sistem informatic pentru generarea automată a programelor 'NC', destinate prelucrării prin frezare a suprafețelor de geometrie complexă, utilizând algoritmi proprii de generare şi simulare a traseelor de prelucrare.

The existence of a great diversity of commercial CAD/CAM products, having equivalent usage features, generates difficulties for the companies in order to choose the most suitable product for theirs specific needs. On the other side, the 'well-known companies' software product proposals require substantial investments for acquisition and nominated users training. Therefore, in the case of the small and medium enterprises, it is recommended the development of customized solutions for automatic generated NC programs, using shareware graphical libraries like 'Open' type (OpenGL, OpenCascade) and a proper programming language for graphical applications. On the ground of this major objective, this study is proposing an informatic system for automatic generating NC programs. These programs are meant to complex surfaces milling, using customized algorithms for tool path generation and simulation.

Keywords: numerical command; packet diagram; class diagram; tool path; graphical library; programming language.

1. Introduction

The existence of a great diversity of commercial CAD/CAM products, having equivalent usage features, generates difficulties for the companies in order to choose the most fitted product for their specific needs. This issue disposes numerous studies and researches, performed at world-wide level.

[1] Prof., Dep. of Manufacturing, "Politehnica" University of Bucharest, ROMANIA
[2] Eng., Geci Engineering Bucharest, ROMANIA

CETIM magazine (Centre Technique des Industries Mecanique) has published a study which brings up and compares, from technical and performances specifications point of view, 11 most used CAD/CAM systems from the industry [1]: CATIA, CAMeLOT, EUCLID MACHINIST, GOELAN, HYPERMILL, POWERMILL, TEBIS, UNIGRAPHICS, VISI-CAM, WORK-NC, SURFCAM.

Having similar features series, almost all of the above software systems provide 2D drafting, 3D modeling or 3D surface modeling facilities. For surface development there are available advanced functions like Bezier, B-Spline or NURBS to generate the interpolation, and again the 3D modeling could be parametrical or variational.

Regarding the manufacturing side, the programmer has available an important technological data base, including machine-tools, tools and process rating parameters.

Most of all the above mentioned systems have, also, robotic programming modules. The models for data exchange between different software systems are using special interfaces like IGES, DXF, STEP or STL [2,3,4].

All CAD/CAM systems listed above, require substantial investments for acquisition and nominated users training. Therefore, in the case of the small and medium enterprises, it is recommended the development of customized solutions for automatic generated NC programs, using shareware graphical libraries like ‘Open’ type (OpenGL, OpenCascade) and a proper programming language for graphical applications, like C++.

In compliance with the above paragraphs, the major target in this work will become the development of automatically generated *NC* programs application, for milling manufacturing of complex geometrical shapes, using personalized algorithms for tool path computation or simulation [5]. In the development process it will be used the OpenCascade graphical library shareware environment and Visual C++ programming language [8]. The final product will try to achieve the following purposes:

- definition of an interactive graphical context for the diversity of geometrical shapes visualization;
- automated generation of numerical command programs for 3 axes milling, needed at rough cutting and finishing operations from manufacturing process applied to the geometrical shapes already loaded into the graphical context;
- tool paths simulation, for the inside instructions of generated *NC* programs.

The first stage in application development is to draw up the static diagrams for system internal structure description. This kind of description is made by identification of: application classes, attributes and operations, and also the relationship between classes. Static diagrams are constituted from:

- package (module) diagram: it shows the application classes groups and relations between them; each package contains related classes;
- class diagram: it shows all the classes that compose the application and the relationship between them.

The diagram process elaboration must include all the concepts that participate to the accomplishment of the anterior objectives:

- to give the means for 3D geometrical models with complex shape to be loaded inside of an interactive graphical context;
- to give the methods (functions) to automatically generate numerical command programs for 3 axes milling manufacturing of already loaded parts.

Discovering the concepts, the relationships and interactions between them, will finally carry out the informational data flow from the application running phase. The analysis presented in this study is made using UML (Unified Modeling Language) notions, which are usually used at object oriented application development [7].

2. Application class packages diagram

All of the application components can be grouped into object classes standalone packages, which can interact with each other by instancing (appealing the class with 'new' operator, or class data member initialization with specific values, generates an object class instance). The developed informatic system main components call a set of concepts from inside of development environment software packages, such as: Visual C++ with the object library MFC (Microsoft Foundation Class), can be fit in MFC package; OpenCascade's graphical objects library, can be fit in OpenCascade package; OpenGL's graphical objects library, can be fit in OpenGL package.

These three big components can be considered like specific packages owned by the application development environment. These components assure several computing functions, needed for all specific application software components. In *Fig. 1* is presented a graphical display for the application package diagram.

According to the diagram and, also, specific to the developed informatic system (named 'Milling'), there are present the following application component packages:

- Milling – contains the application main classes;
- Simulation – contains the simulation process classes;
- Auxiliary – contains classes for application messages view.

All the *Milling* application components (classes) are also included inside of above class packages (*Milling*, *Simulation* and *Auxiliary*). Between these there are interdependence relations through contents and instances. The application type, from Visual C++ point of view, is a MDI project (*Multiple Document Interface*), and the project's software structure is elaborated in conformity with 'document-view' architecture.

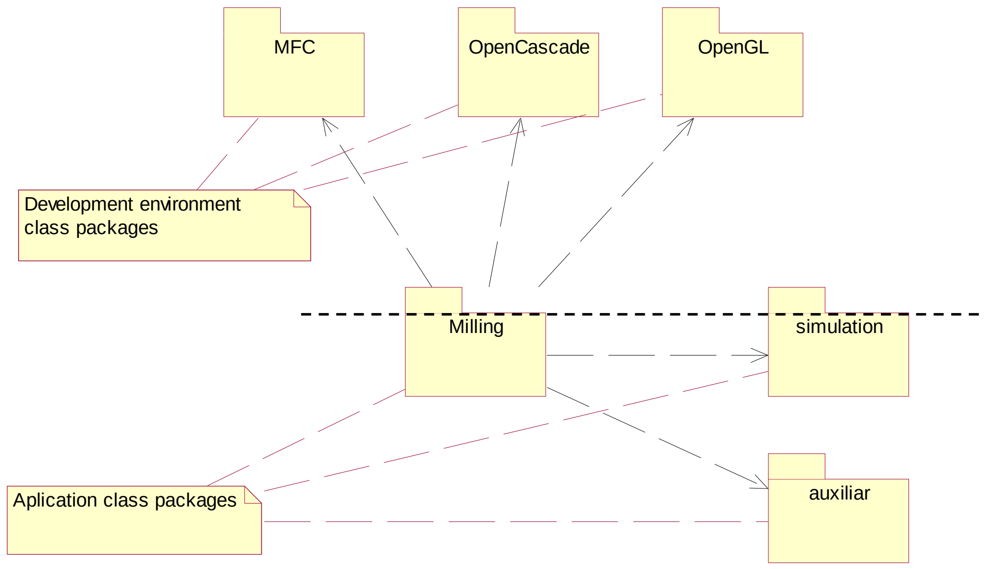

MFC
OpenCascade
OpenGL
Development environment class packages
Milling
simulation
auxiliar
Aplication class packages

Fig. 1. Application class packages diagram

CMillingApp is the class which launches the application in execution and, also, manages the application interaction with the operating system. This application component is extended from *CWinApp* MFC's standard class. Following this pattern, *Fig. 2* shows the application general class diagram elaborated for the 'document-view' like interaction structure type. Resulted application class diagram shows all the four main typical concepts for this kind of software architecture: document class, visualization class, frame class and application class.

3. Application class diagram

Inside Milling package it can be found all the application components, which are transformed into classes instance (objects) and, also, are managed by the application class extended from *CWinApp*.

The Table 1 shows the application components names, and, also, the suitable program source file names from the application sources distribution.

Table 1

Diagram class components

Package	Component (class)	Source file (.h , .cpp)
Milling	CMillingApp	Milling
	CChildFrame3D	ChildFrm3D
	CMillingView3D	MillingView3D
	CMillingDoc	MillingDoc
	CMainFrame	MainFrm
	CMessageView	MessageView
	CDialogDegrosare	DialogDegrosare
	CDialogFinisare	DialogFinisare
	CDialogNCSimulare	DialogNCSimulare
Simulation	NCprog	NCprog
	NCinstr	NCinstr
	CutterGeom	CutterGeom
	ToolShape	ToolShape
	CPostPro	CPostPro
	ViewManager	ViewManager
	CONSTANTS	CONSTANTS
Auxiliar	CsizingControlBarCF	scbarcf
	CsizingControlBarG	scbarg

The general application class assembly and the interaction between them have a simplification description in *Fig. 2*. The figure diagram brings up all the relationship between concepts and, also, their specific attributes (class data and operations members). The class linking relations assure collaboration possibility among classes instance (objects) in application running time.

This class diagram shows three main relation types:

- **dependency**: when a class call another class instance (object) attributes, or in the class operation will be created instances (objects) for another class; this type of link is used when the modification of class behaviour has as result the modification of the behaviour for another class;
- **association**: when a class have another class like attribute types, or have a group of attributes data with another class type;
- **extension** (generalization): when a class expanded from a parent class, and inherits all the parent class attributes; in C++ it is possible that more than one class to be extend from the same parent class.

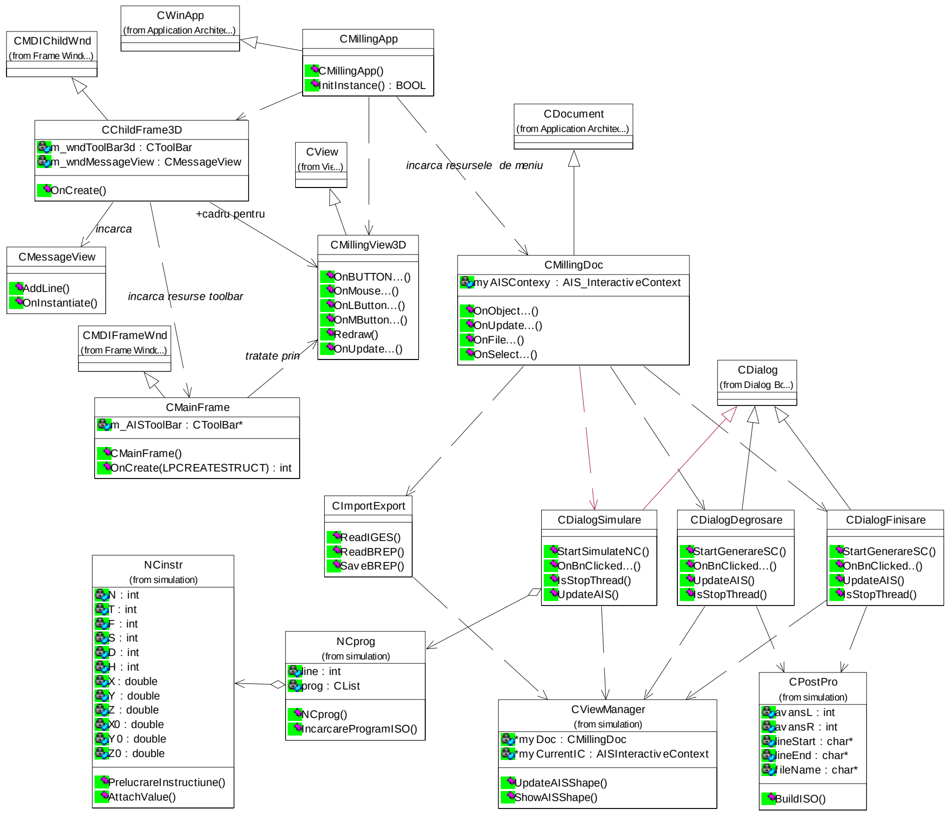


Fig. 2. Application main class diagram.

The class diagram elaboration for each of the application components package will allow to understand the object instance from any class inside the package. The class group that builds up the 'document-view' project's structure architecture is shown inside the class diagram from *Fig. 3*. The class document role is taken by *CMillingDoc* and this class manages the program data. The visualization class, which has the function to manage the interaction between user and document is assured by *CMillingView3D*. The frame class, which includes visualization context and user interface elements, is defined by the *CChildFrame3D* class structure. The last one, *CMillingApp*, takes the role of the application class, which have to launch the application in running stage.

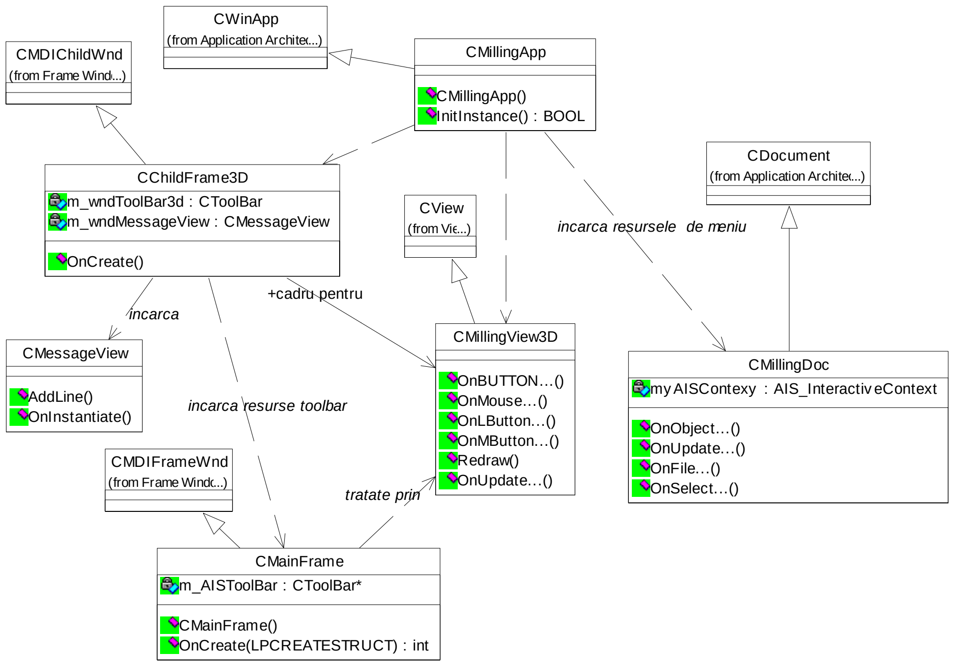
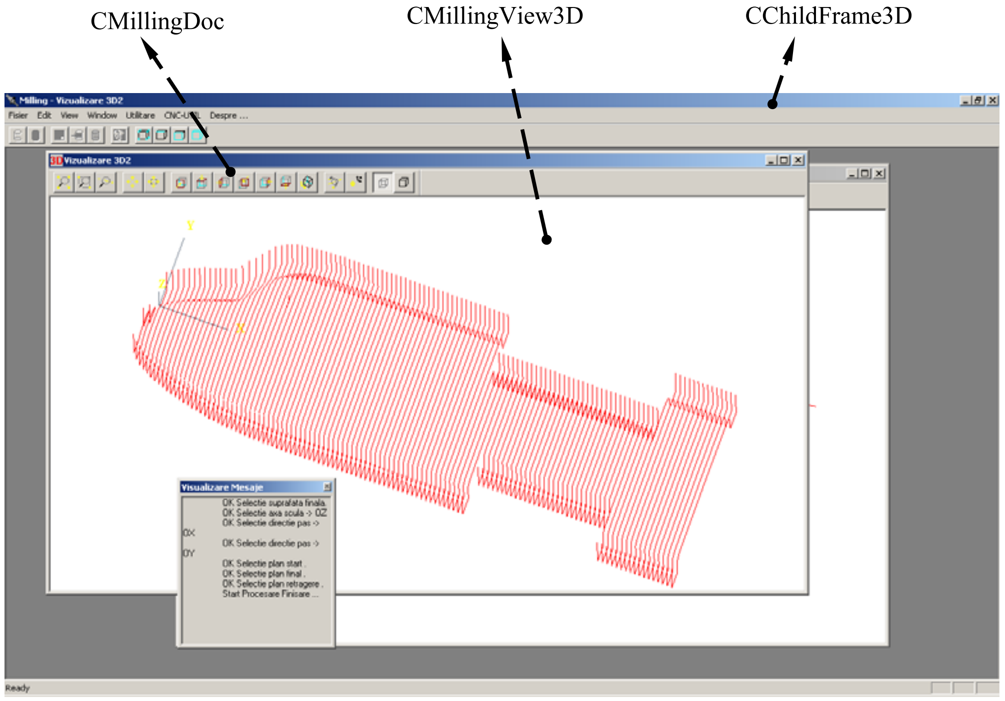


Fig. 3 ‘Milling’ package class diagram.

Once the application is started in execution (*Fig. 4*), it will be created a CMillingApp class instance (which inherit *CWinApp* MFC’s class). The generated object will begin the application main classes calling process, and so the resulted instances will perform the interaction between operating system, application functions and user. Like in the above paragraph, the main classes are: *CChildFrame3D* (inherit CMDIChildWind), *CMillingView3D* (inherit CView) and *CMillingDoc* (inherit CDocument). The class *CMillingApp* loads all the application main window menu resources, and the menu options selection events are solved inside the document class *CMillingDoc*.

Through *CChildFrame3D* there are loaded the toolbars (*Fig. 4*). Toolbars option selection events are solved by the visualization function (operation) class members. Through *CChildFrame3D* there will be generated the application frame needed for *CMillingView3D* instance, which will give the operations required to answer at the main window toolbar selection events (*Fig. 4*). All the events, which manage the user application interaction through the menu or toolbar option selection, are solved by calling the *CMillingDoc* and *CMillingView3D* class instances. The last one object (CMillingView3D) has also the role to ask on the events generated by the: mouse movement, mouse buttons pushing or keyboard buttons pushing. The *CMessageView* class assures the messages sent by the application functions to be shown in the text context window, placed on the bottom side of the main application frame.



```
OK Selectie suprafata finala.
OK Selectie axa scula -> OZ
OK Selectie directie pas ->
OX
OK Selectie directie pas ->
OY
OK Selectie plan start .
OK Selectie plan final .
OK Selectie plan retragere .
Start Procesare Finisare ...
```

Fig. 4 The '*document-view*' architecture for *Milling* application.

Another class assembly is the one grouped inside the 'Simulation' package. The class attributes included there assures the numerical command (*NC*) program instructions reading and parsing process, for each sequence line. The class diagram for this package is displayed in the *Fig. 5*.

The parsing process for only one *NC* program sequence line is made within the *NCinstr* class. Inside this class frame there are class operation members, which read the *NC* program line and extract the tool centre geometrical position coordinates, and also the type of manufacturing movement.

The *NCprog* class is composed of *NCinstr* type items collection. This items collection is a simple linked list type *CList* (structure type from MFC package). Therefore, it can be considered that *NCprog* class is an aggregate of *NCinstr* class.

The *CONSTANTS* class contains static data members, with constant data values, which are used in the computing process, by a variety of functions from application classes.

The *ToolShape* class structure has methods for tool path segment computation, starting from the *NC* program successive instructions, and also has methods for building of the geometrical shape generated by tool movement along the tool path segment, described by *NC* sequence instruction.

The *CPostPro* class structure has the needed methods in writing the resulted output *NC* program file. For a specific milling operation, this processing phase is started after the end of the tool path generating process.

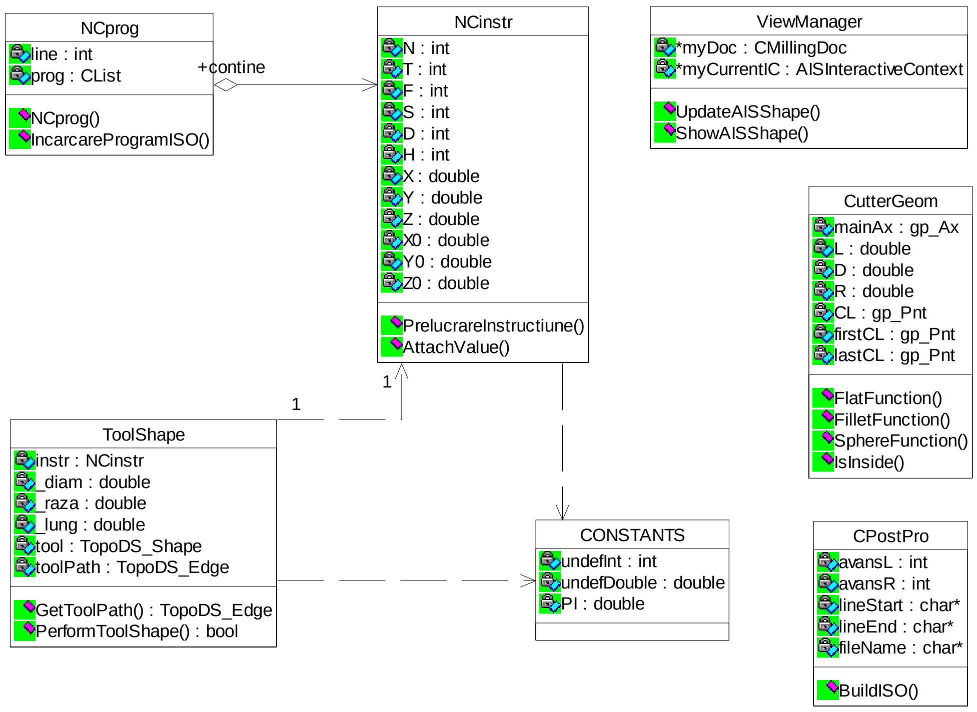


Fig. 5 ‘Simulation’ package class diagram.

Inside *Fig. 6* it is shown the class diagram, requested by the system, to accomplish the user application interface, through the application main frame menu options. Basically, the menu is a command messages list, which can be selected and send to the active window frame. The menu is associated to the application main window frame. Each menu item option requests a particular class member operation, contained by the document class. The description of these operations, will show the programming mechanism used to achive the interaction between user and application at the menu level. Throught these command messages sent to the application it will start running also other different user interface types, named dialog box. This kind of user interfaces allow to collect multiple input data, of various data types, allowing input data validation at input or choosing it from predeterminate data lists.

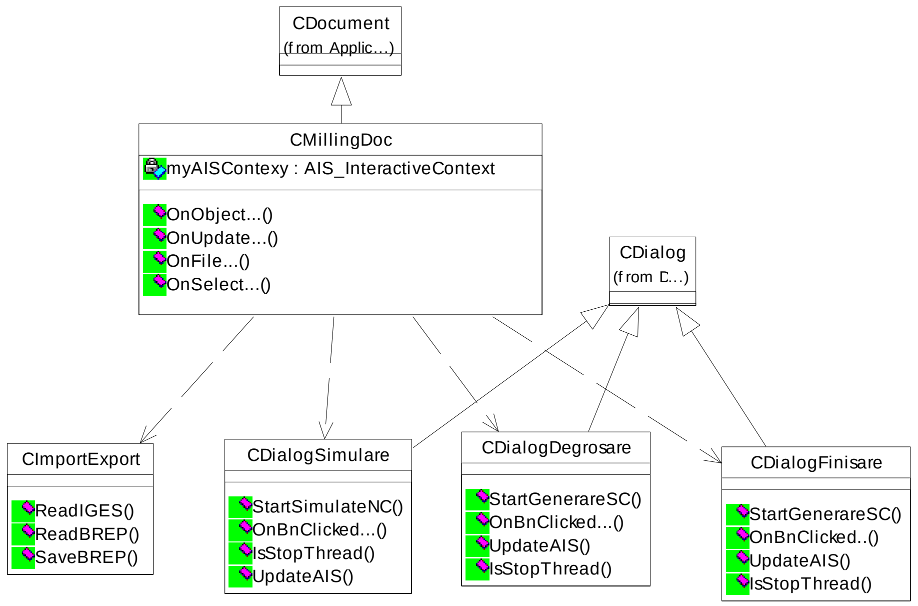


Fig. 6 Main menu options class diagram.

The above diagram presents these structures data type, extended from CDialog MFC class, which is met into the following classes: *CDialogDegrosare*, *CDialogFinisare* and *CDialogSimulare*. These classes launch the tool path generation computing process, for specific milling operation, using the '*execution thread*' programming method [8, 9].

The '*working function*' from inside the execution thread, used within the class, follows an alghorithm rule illustrated in the logical scheme displayed in *Fig. 7*.

The output of the processing made inside these functions, is a segment collection, ordered and oriented, which is generated starting from the intersection curve group between part's free form and the machining planes. This segment collection is postprocessed in order to obtain the appropriate numerical command program. The '*working function*' processing intrerruption or termination will save the generated segments assembly as output, using a *BREP* format file.

The *CImportExport* class assures the methods for *IGES* or *BREP* file format reading, and also the saving of the geometrical objects using *BREP* file format. These methods give the possibility to import free forms into the application interactive graphical context, taken from different CAD systems.

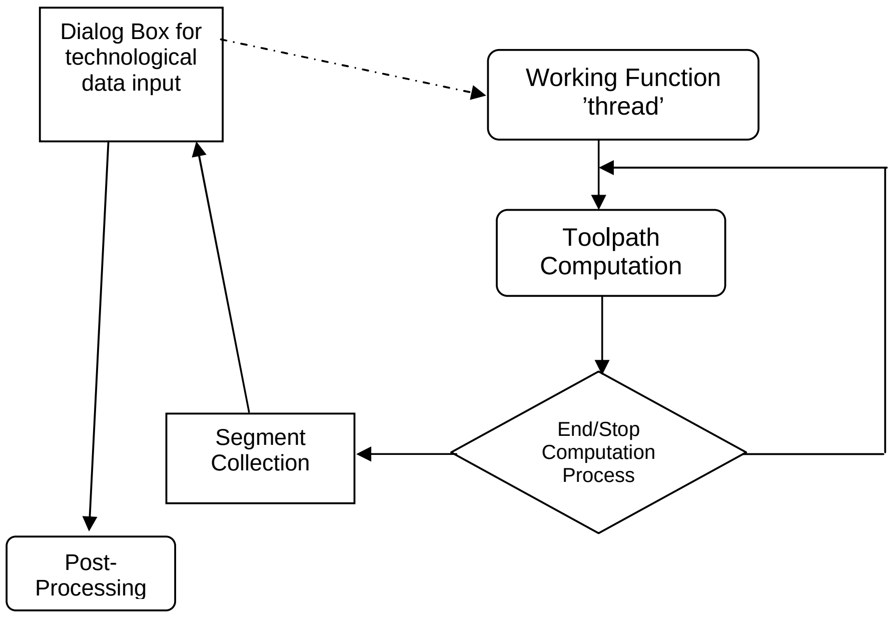

Dialog Box for technological data input
Working Function 'thread'
Toolpath Computation
End/Stop Computation Process
Segment Collection
Post-Processing

Fig. 7 Logical scheme follow by the functions for tool path computation.

4. Conclusions

The development of an informatic system intended to automatically generate numerical programs for the 2.5 or 3 axis milling process, becomes an available activity by using of free graphical objects library like OpenCascade or OpenGL. This activity can be a viable solution for small and medium companies, which have a low or insignificant budget allocated for the numerical command development of manufacturing department. The application, that was illustrated in this study, intends to answer to the following particular issues, like any other CAM application:

- self-determining to any CAD solution, by availability of *IGES* file format importing;
- availability of a graphical interactive context, for geometrical objects visualization and manipulation;
- functions for numerical command generation needed in rough cutting and finished milling;

- function for resulted tool path simulation.

The application development can be made in compliance with the existing machine tools central unit particularities, and also in conformity with the more often used milling operation types (contouring, cavity or boss manufacturing, and so on).

B I B L I O G R A P H Y

[1]. *Zapciu M.*, Fabricaţia asistată de calculator, Editura POLITEHNICA PRESS, Bucureşti, 2003

[2]. *Ivan N. V., ş.a.*, Sisteme CAD/CAP/CAM. Teorie şi practică., Editura Tehnică, Bucureşti, 2004

[3]. *Machover, C.,* The CAD/CAM handbook, McGraw Hill, 1996.

[4]. *Udroiu., R.,* Concepţia şi fabricaţia pieselor de formă complexă., Teza de doctorat, 2003, Conducător ştiinţific: Prof. dr. ing. N.V. Ivan

[5]. *Condei., V.,* Contribuţii la generarea automată a programelor CNC pentru prelucrarea prin frezare a suprafeţelor complexe., Referat nr.3 la teza de doctorat, UPB 2006, Conducător ştiinţific: Prof. dr. ing. C. Neagu

[6]. *** – OpenCascade, http://www.opencascade.com/pub/doc/OCC62_Overview.pdf , 2007.

[7]. *Bell,D.,*UML Basics, IBM Rational Software, http://www.ibm.com/developerworks/rational , 2003.

[8]. *Williams, B.,* Bazele Visual C++, Editura Teora, 1997.

[9]. ****,* -MSDN Library, http://msdn2.microsoft.com/en-us/library/69644x60(VS.80).aspx, 2007.