Computer Science Journal of Moldova, vol.3, no.2(8), 1995

The Bergman package shell: an example of interface to the interpreting system*

A.Colesnicov L.Malahova

Abstract

Engineering problems of creating a conversational shell for an interpreter are discussed using as the example the shell for the computational algebra package Bergman. The attention is focused on selection of the program objects, their interaction and reusage.

1 Introduction

There is a lot of popular and frequently used batch or teletype-dialog programs which were embellished by attaching to them corresponding shells. Restricting ourselves by MS-DOS systems only, we have DosShell, Norton Commander (and many others) for MS-DOS, Shez (and many others) for archivers, TEXshell and Scientific Word for TEX, etc. All these shells have much common in their external presentation and behavior. It may be supposed that their internal organizations may equally have much in common. These engineering aspects are discussed in the article.

We will use as the example a shell for the computational algebra package called Bergman [3, 5, 6]. Bergman was developed in the Stockholm University, Sweden, by J.Backelin to calculate the Gröbner basis in the commutative and non-commutative cases. The package was implemented in the Portable Standard Lisp (PSL), firstly under UNIX. Then efforts were applied to further develop the implementation in the

©1995 by A.Colesnicov, L.Malahova

*This work was partially supported by the Royal Swedish Academy of Sciences, project number 1502.

cooperative project with the Academy of Sciences of Moldova. The package was successfully transferred to MS-DOS. Now the project is continuing.

The Bergman package has three function of the uppermost level which perform typical calculations in three cases. These three function are composed from about of twenty function of the lower level. In some cases the user may wish to compose his own upper level function from those twenty. There is a lot of internal variables and flags whose values control modes of the calculation.

The used software, Reduce+PSL, is a teletype-dialog system for all platforms. So, to perform a calculation with Bergman, the user is not only to prepare the source data, but to type manually all necessary flags and variables assignments, then, possibly, to enter a new upper level function definition, and, finally, to call the function to solve his problem.

In MS-DOS case, the work was partially automatized with the Bergman shell. Using the shell, the user has advantages of the intuitive and simple interface, when he clicks buttons and check boxes instead of entering LISP commands (see Sec. 2.3 below). The DOS-Lisp interaction was implemented in the first shell version through disk files. It was the compromise solution chosen for its simplicity. With it, the possibility to compose own procedures from the intermediate level functions still exists (see Sec. 2.3, page 116 on the `Task: Run batch` menu item). But the user has no possibility to interact with the system during the LISP calculations.

The solved problems and the generalizations are discussed below.

Another example of an interpreter shell may be found in [2].

2 The presentation and behavior of the Bergman shell

2.1 The shell running loop

The Bergman shell is launched from the PSL interpreter. The sequence is as follows:

1. The Reduce interpreter is started consuming a standard script as the input program. This input script contains Reduce commands switching to PSL mode, the LISP function `DOSSHELL` definition and the infinite LISP loop calling this function.

2. The PSL calls `DOSSHELL`.

3. Through the MS-DOS PSL extension – the `SYSTEM` function – the MS-DOS shell executive module `BERGM_SH.EXE` is started under MS-DOS. It is important that in the moment the PSL interpreter, the compiled image of the BERGMAN package and all memory allocated for them is used. We have MS-DOS and the Bergman shell in the lower memory and the package waiting in the upper (extended) memory. For all shell operation we have no more than 400 Kbytes.

4. The shell is utilized by the user to prepare LISP input files with function calls, and data files with polynomials.

5. The shell finishes its work writing prepared files to the disk and rerouting the PSL interpreter input from that file containing function calls.

6. The interpreter interprets newly created file and calls functions solving the algebraic task. The LISP teletype-mode output is seen on screen.

7. The interpreter input from files is nested. Finishing the algebraic calculations, it returns to the infinite loop calling shell (step 2).

8. To break the loop, the user may, within the shell, press the `Alt`+`X` key combination or to click with the mouse the `Exit` menu item. The shell writes to the disk the program of the single LISP command '`(QUIT)`'. This command is interpreted on step 6 and forces the loop break by simply stopping the interpreter. (During the LISP interpretation, the user may press `Ctrl`+`C` to stop.)

2.2 Usual menu items

The Bergman shell was implemented using Borland Pascal with Objects 7.01 and its supplied package Turbo Vision [4]. Its external representation is usual for such programs.

You see on the screen the shell desktop between the menu bar at the top and the status line at the bottom. The menu items may be selected by keys or mouse. We have six usual menu items (`File`, `Edit`, `Search`, `Options`, `Window`, and `Help`), and three specific items (`Task`, `View`, and `Additions`).

Through the `File` menu item we can open files for editing, saves files, change the current directory, exit temporarily to MS-DOS, etc. The `File: Exit` item means writing the LISP program '`(QUIT)`' as the next interpreter input before terminating the shell (see step 8 above). The exit from the shell to LISP calculations (not stopping the interpreter) is performed from another item (see below).

We had used a standard Turbo Vision implementation of the `File` item. Our experience shows that in many cases the desired extension to this submenu is `File: Erase` to delete a file from the disk.

Other usual menu items do not need any comment.

2.3 Specific menu items and their functions

To define the solved task, the user is to pass three menu items – `Task: Options`, `Task: Flags`, and `Task: In/Out`.

When the user selects the `Task: Options` menu item, he sees the Options dialog. This dialog contains the calculation defining panel – the Commutativity radio buttons, the Strategy radio buttons, and the Poincaré-Betti and Hilbert series check box. Combining these setting, the user selects one of the three possible calculations:

`Simple` Commutative or non-commutative algebra, ordinary Buchberger's algorithm, no Poincaré-Betti and Hilbert series.

`StagSimple` Commutative algebra, staggered linear basis algorithm with substance (SAWS), no Poincaré-Betti and Hilbert series.

`NcPBH` Non-commutative algebra, Buchberger's strategy, double Poincaré-Betti and Hilbert series of the associated monomial ring calculation.

For the `Simple` calculation the user is to select the Ordinary Strategy not checking PBH Series box. He can select any Commutativity – commutative with lexicographical order, commutative with reverse lexicographical order, and non-commutative.

For the `StagSimple` calculation the user is to select the SAWS Strategy, commutative (lex or rev-lex) Commutativity not checking PBH Series box.

For the `NcPBH` calculation the user is to select the Ordinary Strategy, non-commutative Commutativity, checking PBH Series box.

The dialog automatically blocks illegal calculation option combinations (see Sec.4 below). To change previously set options, the user may need to mark them in some order. E.g., if he had solved the `StagSimple` problem before, he had marked commutative (lex or rev-lex) and SAWS buttons, not marking PBH Series box. In the case the non-commutative button and PBH Series box are disabled, and the user can not mark them. To select the non-commutative variant, he marks at first the Ordinary Strategy – then non-commutative button will be enabled (but PBH Series box remains disabled until the user does select the non-commutative radio button).

The selected calculation mode is indicated in the lower-right corner of the screen (in the status line).

Other elements of the `Task: Options` dialog are independent and may be selected in any order.

The `Task: Flags` dialog permits to define desired flags. Depending of the calculation mode, some flags may be disabled. Except specific, there are non-specific flags, e.g., if the user wants verbose garbage collection, he marks `Verbose GC` checking box. It corresponds to the standard PSL flag '`GC`'.

The last dialog is the `Task: In/Out` dialog where the user points to or enters input and output information. Depending of the calculation mode, there are three different dialogs – `Simple` defines only one output result, `StagSimple` – two, and `NcPBH` – three. Results may be

directed to the file, or to the screen. The input may be written directly on the screen, or taken from the file. To define this the user selects the corresponding button and then presses the [→] key, or clicks the onscreen → with the mouse.

In any file selection dialog the user has the possibility to open file for editing. Closing the corresponding edit window he returns to the same file selection dialog.

Having selected the input and output modes and entering data the user can press [`Enter`] of click the Run button in the `Task: In/Out` dialog to start the Bergman calculation. It means that the shell writes on the disk the LISP program. E.g., the program contains a line '`(ON flag_name)`' or '`(OFF flag_name)`' for each flag `flag_name` meaningful for the chosen calculation mode, depending of the settings in the `Task: Flags` dialog. Then the shell reassigns the PSL input from the formed file and terminates.

When the calculation is finished and the shell is reentered, the user can view the LISP echo screen pressing [`Alt`]+[`F5`], and view the output results through the `View` menu item.

The user can prepare the whole task in a single file and run it through the `Task: Run batch` menu item.

The `Additions` menu item calls additional programs not included in the Bergman package. One such program is the Anick's resolution calculation [1]. The algorithm was developed by V.Ufnarovski and implemented in C++ by postgraduate student A.Podoplelov. It is the test variant which had shown satisfactory results (see [6]) and is supposed to be rewritten in LISP to be included into the Bergman package itself. The rewriting would enlarge available memory, would permit to use intermediate results and the Gröbner basis from the LISP memory, and has other advantages.

The Anick's resolution program has its own parameters and command language. The `Additions: Anick's resolution` dialog has an additional window showing the formed command sequence whilst the user clicks buttons and fills input fields. E.g., if the user presses the `Derive all chains` button and then fills the input order field with the number 5, he sees in the command sequence the '`d 5`' command added

to calculate derivations of order 5 chains. The user may edit the formed command sequence directly in its window. This property was found a very useful and demonstrative one so it is planned to implement such window for the main formed LISP program.

3 Object classes for Bergman shell

The shell is a Turbo Vision application and has many commonplace objects as such. We discuss below the unusual ones.

3.1 Configuration and task

There is two kinds of shell status information to keep between two subsequent shell executive module calls: the configuration and the task.

The configuration contains almost constant data like the color palette[1] chosen by the user in the `Options: Palette...` dialog. The configuration is stored in the `BERGM_SH.CFG` file.

The current solved task defines the file names, flags and options settings etc. It is to be kept in the file because the Bergman shell would be terminated to the moment of algebraic calculations. When the shell is recalled, the task is restored, all opened dialogs will be filled with the corresponding data, and the user may change these settings. The task is stored in the `BERGM_SH.INI` file.

The usual way to implement the status information keeping is to write to the file and to read from it a lot of variable values. In the shell all status defining variables are fields of two objects of `TConfiguration` and `TTask` classes. These objects have methods to load and to store themselves and to reset system status.

In fact both these object classes are inherited from one which has the following base properties of configuration and status objects:

1. In constructor, the object searches for the configuration file. If the corresponding file is not found, it calls its abstract `SetDefaults` method and creates a new configuration file.

[1] Three possibilities are color, black/white, and monochrome (LCD).

2. The object deals with the configuration file signature containing the version number. If the status file layout is of the lower version, the object calls its abstract `ProcessLowerVersionData` method, and then rewrites the configuration file.

3.2 Extendable clusters

In Turbo Vision, `TCluster` is the abstract object class from which radio buttons and check boxes are inherited. The extendable cluster is the modification of the standard cluster which informs its owner on changing its status (switching a radio button, checking or unchecking a box). It is used to simultaneously change the associated objects status, see Sec. 4 below. The property is desirable for all objects, it would be great to made it the generic property of all Turbo Vision views adding it to `TView` class.

3.3 Radio button extender

The radio button extender is a new object which can be associated with a particular radio button from the extendable cluster of radio buttons. The association is alike to that exiting in Turbo Vision between input lines and history windows, and is described in details in Sec. 4.2 below.

4 The Bergman shell objects interaction

There are many cases in object-oriented design when objects are not totally independent and the common communication method trough messages is not sufficient or inconvenient. In the Bergman shell we had several examples of the situation with different dependency grade.

4.1 Mutually blocking clusters

The external behavior was described in Sec. 2.3. Three calculation modes are selected by setting combinations of a cluster with 3 radio buttons, a cluster with 2 radio buttons and a check box. There are 12 possible combinations; 6 combinations are impossible, 3 combinations

define the `Simple` mode, 2 combinations define the `StagSimple` mode, 1 combination defines the `NcPBH` mode. May be it would be simpler to list possible combinations in a cluster of 6 radio buttons, but the current design reflects algebraic realities.

The problem of blocking the unpermitted combinations was solved in the `HandleEvent` method of the `Task: Options` dialog which contains these clusters. The corresponding object class contains also the special method `Dependencies` and the special Boolean field `Starting`. The `Dependencies` method's argument is an integer code informing that the Commutativity, the Strategy, or the Poincaré-Betti and Hilbert series request is changed.

In the `Init` constructor, the `Starting` field becomes true, and cluster status is restored from the current task. In the `HandleEvent` method the `Starting` flag is tested. If it is `True`, the `Dependencies` method is called thrice with three possible arguments blocking undesired item in any case, and the `Starting` flag is set to `False`. This initiating sequence is executed only once, and its success strongly depends of the fact that the previously set and restored combination is permitted. The "most initial" combination is set as default when the task file does not exist, and it is a permitted one.

Three calculation mode clusters are extendable clusters (see Sec. 3.2 above). Being changed, each of them send a message to its owner, the `Task: Options` dialog. The dialog's `HandleEvent` method simply calls the `Dependencies` method with the corresponding code.

The `Dependencies` method is straightforward. For example, if Commutativity was changed, it checks the current commutativity; if it is the commutative case, then it have changed to commutative right now; so the method locks the Series check box and unlocks the SAWS strategy, etc.

4.2 Radio button extenders at work

Three `In/Out` dialogs include the clusters shown on Fig. 1.

They work as follows: if the user selects the `File` radio button, then its extender (you see it as $\rightarrow$) became active and may be clicked,

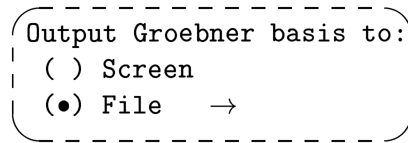

Figure 1:Extended radio button (part of `In/Out` dialog).

or the user can press the [→] key. Moreover, the user can click the extender, and then the corresponding radio button becomes selected. There can be several extenders on the screen, but only that one reacts to the [→] key whose radio button is currently selected (and marked).

The clicked extender shows the standard file dialog to select the file name. After the dialog is closed, the selected file name is shown below the radio button. This is analog to the combo boxes known in MS Windows and other systems.

The extender became active after the corresponding radio button because the last one is extendable and issues the message when changed. If the message code is recognized by the extender, and the button cluster value (button number) equal to that attached to the extender, the extender becomes active. If the radio button cluster is switched to another button, the extender becomes inactive, it can not be clicked and does not react to the [→] key.

The extender keeps the pointer to the associated cluster. If it is clicked, it activates the cluster and selects the corresponding button.

The desired property of visual object is this possibility to be associated. It would be great to include the corresponding fields and abstract methods in the generic `TView` class.

5 Conclusions

The Bergman shell design posed many interesting engineering problems and permitted to formulate several requirements to the shell object classes and to generic classes.

- `File:Erase...` is the desired extension for the `File` submenu.
- The possibility to open file for editing is desired in any file selection dialog, with return to the same file selection dialog when closing this new window. Thus, such windows are to keep information on desktop status at their opening. The "quick view" window to show partially the selected file contents may be also useful for file selection.
- If the shell forms data for the system, it is desired to show the formed data in the separate window, and, in some cases, to permit the user to edit them directly. But this may imply the implementation of the data analyzer to keep the system status in correspondence with the manually edited data. (It is one of the forms of the incremental editing principle – at each step the data are to be correct.)
- It was found that the status and configuration information are to be kept in special objects. An implementation was proposed based on a generic configuration class.
- We found that in many cases objects are more or less associated. Several methods of the coordination were discussed, including that implemented in the owning object and that through the messages transfer and direct links. It was found desirable to preview the possibility of association starting from the most general level of object classes inheritance.

The designed object classes are reusable and proved their usefulness in many cases.

Acknowledgments

Authors thank J.Backelin, S.Cojocaru, J.-E.Roos, and V.Ufnarovski for their help and fruitful discussions. S.Cojocaru had applied invaluable efforts as alpha-tester of the Bergman shell.

References

[1] D.Anick. On the homology of associative algebras.
Trans. Am. Math. Soc., **v. 296**, nr. 2, 1986, pp. 87–112.

[2] A.Colesnicov, L.Malahova. A portable interpreter shell and its implementation for a perspective computer.
In: Computer software and programming (Mathematical Investigations, **issue 115**). Chişinău, 1990, pp. 92–96 (in Russian).

[3] J.Backelin, R.Fröberg. How we proved that there are exactly 924 7-roots.
S.M.Watt, ed. Proc. ISAAC'91, ACM, 1991, pp. 103–111.

[4] Borland Pascal with Objects 7.0. Turbo Vision Version 2.0 Programming Guide.
Borland International, Inc., 1992.

[5] J.-E.Roos. A computer-aided study of the graded Lie algebra of a local commutative Nötherian ring.
J. Pure Appl. Algebra, **v. 91**, 1994, pp. 255–315.

[6] S.Cojocaru, V.Ufnarovski. Noncommutative Gröbner basis, Hilbert series, Anick's resolution and BERGMAN under MS-DOS.
Computer Science Journal of Moldova, **v. 3**, nr. 1 (7), 1995, pp. 24–39.

A.Colesnicov, L.Malahova
Institute of Mathematics,
Academy of Sciences of Moldova,
5 Academiei str., Kishinev,
277028, Moldova
phone: (373–2) 738058
e–mail: *kae@math.moldova.su* (21*mal@math.moldova.su*)

Received 3 July, 1995