COBOL SoftBench: An Open Integrated CASE Environment

With the aid of a mouse and a menu-driven interface, COBOL programmers new to the UNIX® operating system can improve their productivity with a tightly integrated toolset that includes an editor, compiler, debugger, profiler, and other software development tools.

by Cheryl Carmichael

Many large companies today are still developing mission critical COBOL applications, and they want tools to help them easily transition their developers to open systems.

In 1991, we recognized that many of our customers were moving MIS applications to the UNIX operating system using C SoftBench and C++ SoftBench. These customers asked for a more business-oriented language—specifically COBOL. In addition, HP-UX* had just been recognized as the leading UNIX system environment for MIS organizations moving from mainframes to open systems.

We heard customer concerns about moving MIS development teams to open systems. They wanted to minimize the loss of developer productivity associated with learning a new operating system and new development languages and rewriting their custom applications.

The COBOL SoftBench family is based on HP MF COBOL, Hewlett-Packard's implementation of Micro Focus COBOL, which is based on technology from Micro Focus, Ltd. COBOL SoftBench leverages a development team's COBOL expertise and protects its investment in COBOL applications.

When we were designing COBOL SoftBench we aimed at novice UNIX-system users. These users minimize their learning curves with the aid of a mouse and a menu-driven interface. Several tools have easy-to-use graphical user interfaces, and common UNIX commands are accessible via system menus.

COBOL SoftBench improves developer productivity with a tightly integrated toolset that includes an editor, compiler, performance analyzer, debugger/animator, static analyzer, file comparator, email, and more. Other items provided by COBOL SoftBench to improve productivity include:

- Graphical views of complex programs
- Dozens of integrated third party tools
- Provisions for developers to integrate their own tools using the SoftBench encapsulator[1]
- The ability to automate repetitive tasks using the SoftBench message connector[2]
- Support for mixed-language development with COBOL/C SoftBench and COBOL/C++ SoftBench.

COBOL SoftBench

Our tightly integrated toolset increases developer productivity. The interactions between the SoftBench programs that make up this toolset are shown in Fig. 1.

- The SoftBench program editor† edits program files, shows the column and line location of the cursor, accesses version control for a file, and performs common UNIX commands on a file.
- The SoftBench program builder† controls compilation and linking of application files, provides views and jumps to source code for compilation errors, creates makefiles, and displays makefile dependency relationships graphically. The program builder also supports embedded CICS (Customer Information Control System) statements and SQL applications.
- The SoftBench COBOL animator uses an enhanced window interface to the Micro Focus COBOL animator to debug

† See reference 3 for more information about these SoftBench tools.

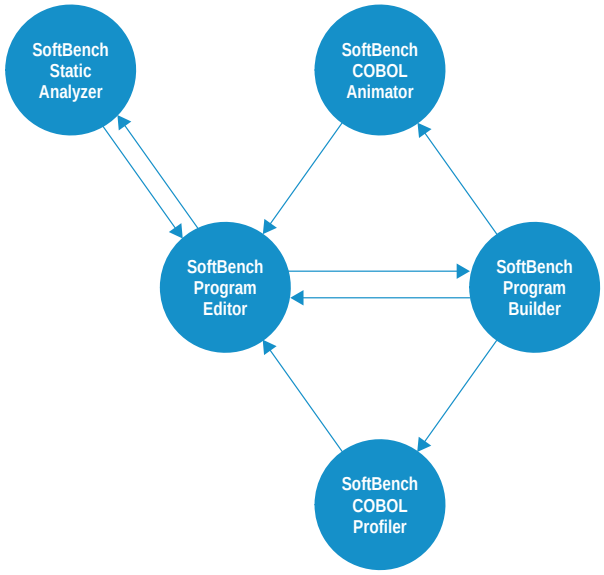

Fig. 1. Interactions between SoftBench tools that make up the COBOL SoftBench toolset.

executable COBOL programs, provide access to native commands for expert Micro Focus users, display data monitors, and allow mixed-language debugging.

- The SoftBench static analyzer enhances visual code browsing by graphical or textual representation of program structure.
- The SoftBench COBOL profiler uses an enhanced window interface to the Micro Focus COBOL profiler to supply detailed statistics about the run-time performance of a COBOL program. It determines performance bottlenecks and helps the developer make changes for the greatest improvements.

Other integrated tools that are not shown in Fig. 1 include:

- The SoftBench development manager† handles file and directory management and version control.
- Computer-based training comes with each SoftBench product. Each computer-based tutorial tells about individual tools and exposes the user to a wide variety of SoftBench capabilities using several hands-on training examples.

Friendly User Interface

COBOL SoftBench developers are buffered from the standard UNIX command line entry format. They select buttons, toggle options, choose menus, and drag-select text. Typing is left for data entry and editing source code. The following model shows how a user would view the UNIX operating system through the COBOL SoftBench interface.

The SoftBench development manager directory list presents a list of files in the currently viewed directory and allows various actions to be performed on any file. For example select Directory: Show Full Listing to view a complete directory listing including permissions, owner, group, size, and modification time (see Fig. 2). The files displayed in the directory list can be limited by using predefined or custom filters.

Within the directory listing, a single line, multiple lines, or even discontiguous lines can be selected for the same action. To select discontiguous files, hold down the **Ctrl** key while clicking the left mouse button on each file name. For example, to compile scattered source files, select the discontiguous .cbl files shown in Fig. 3 and then choose Actions and then one of the compile selections.

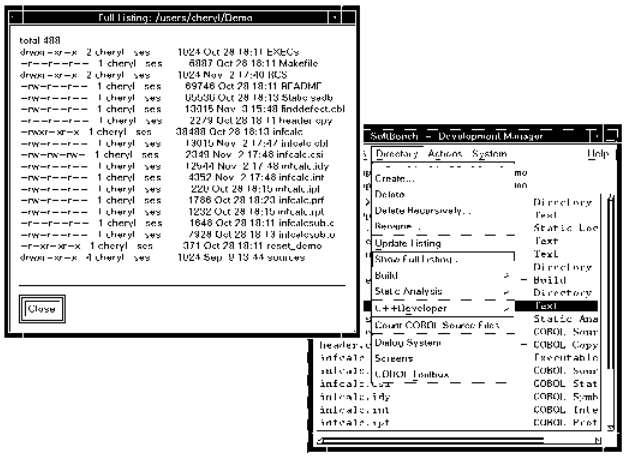

Fig. 2. A directory listing produced by using the SoftBench development manager's Directory: Show Full Listing menu item.

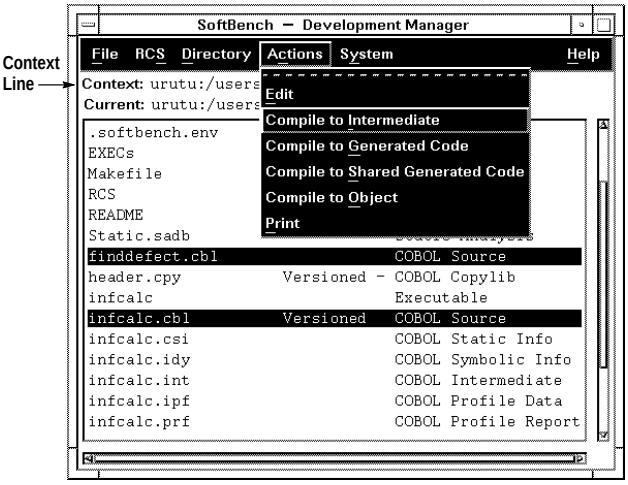

Fig. 3. Selecting several COBOL files for compilation.

Moving about in a directory only requires a mouse movement. To change to another directory and view its contents, move the mouse focus to the context line. While holding down the right mouse button, move through directories until the desired directory is highlighted. Release the mouse button and a new directory list displays.

The SoftBench development manager helps manage files and directories on the developer's computer and across the network. From the File: menu item, the developer can easily choose Edit, Copy, Rename, Import, or Delete depending upon the task.

A number of commonly used UNIX commands are available from the System menu. These include:

- Changing a file's permission
- Printing a file with a number of printer options
- Checking the spelling in a file
- Counting lines, words, and characters in a file
- Creating a backup copy of a file
- Updating the modification time for a file.

Supporting Code Use Model

New COBOL SoftBench users want to leverage their COBOL development expertise and protect their investment in existing COBOL applications. The following example shows how a COBOL developer who is maintaining unfamiliar code would use COBOL SoftBench to tackle a problem that might arise in a COBOL software maintenance environment.

In this scenario the developer receives a complaint that the inflation calculation program (infcalc.cbl) takes too long to start. To deal with this complaint, the developer must quickly identify the performance bottleneck, fix the problem, and verify that the fix does indeed improve the applications performance.

Source Code Control. The inflation calculation program has a fileset containing infc,alcsub.c, infcalc.cbl, header.cpy, and Makefile. The developer sets up a working directory using the SoftBench development manager and checks these files out of a revision control system. The revision control systems that COBOL SoftBench can be configured to use include RCS, SCCS, PVCS, ClearCase, and others.

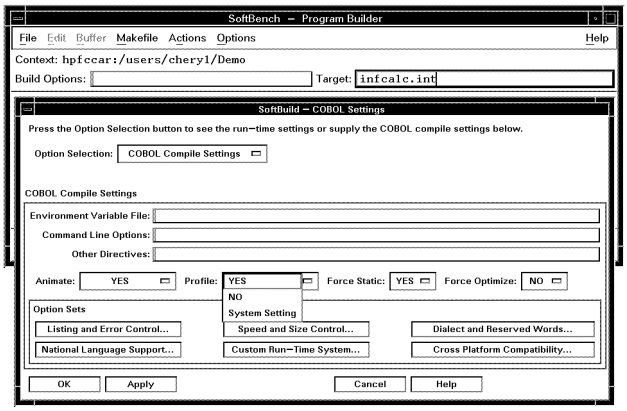

Fig. 4. The SoftBench program builder and the COBOL compile settings dialog box.

Compile Options. The application must be compiled to enable the generation of performance statistics. Using the SoftBench program builder, the developer enables the Profile option with a mouse click in the COBOL compile settings dialog box and compiles the application (Fig. 4). The COBOL SoftBench environment accepts many control options which tailor the compile for special needs.

Micro Focus Run-Time Settings. Over 200 options are available in the Micro Focus COBOL development environment. COBOL SoftBench provides a way to view and change options and values easily. Over 50 of the most common options are available from the SoftBench program builder via the COBOL compiler settings and COBOL run-time settings dialog boxes.

Application Execution. The application is executed to generate performance data. When the developer runs the application from the SoftBench program builder, the files infcalc.rpt and inflcalc.ipf are created.

Application Performance Analysis. The SoftBench COBOL profiler makes it easy to determine where the application is spending its time. Remember that for our example, the performance complaint is about slow startup. The developer notices that a large amount of time is being spent in the initialization module (Fig. 5).

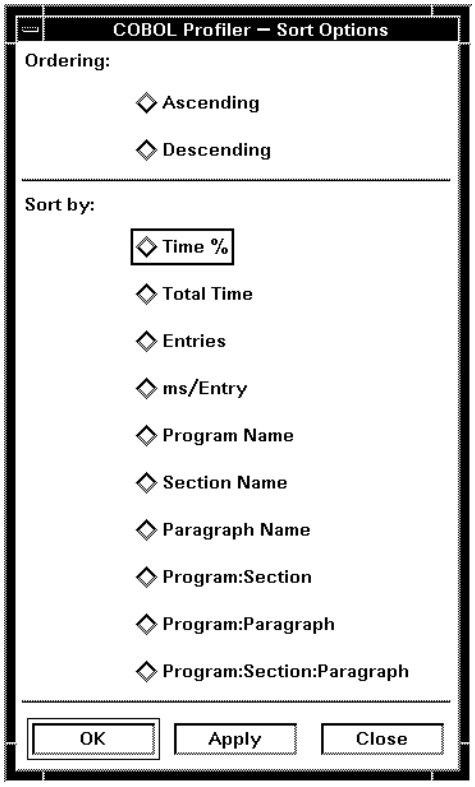

Fig. 6. The ten options for sorting performance statistics. In this example the modules in the program being evaluated will be sorted according to the amount of time spent in each module.

The SoftBench COBOL profiler is an encapsulation of the standard Micro Focus COBOL profiler tool. The COBOL SoftBench implementation enhances the Micro Focus profiler listings of performance data by displaying the data in a SoftBench window. Flexible reporting capabilities are available. For example, there are ten ways to sort performance statistics (see Fig. 6). Using the Options: Display Options from the profiler window, the developer can also choose to view only entered data, nonentered data, or all the paragraphs in the program.

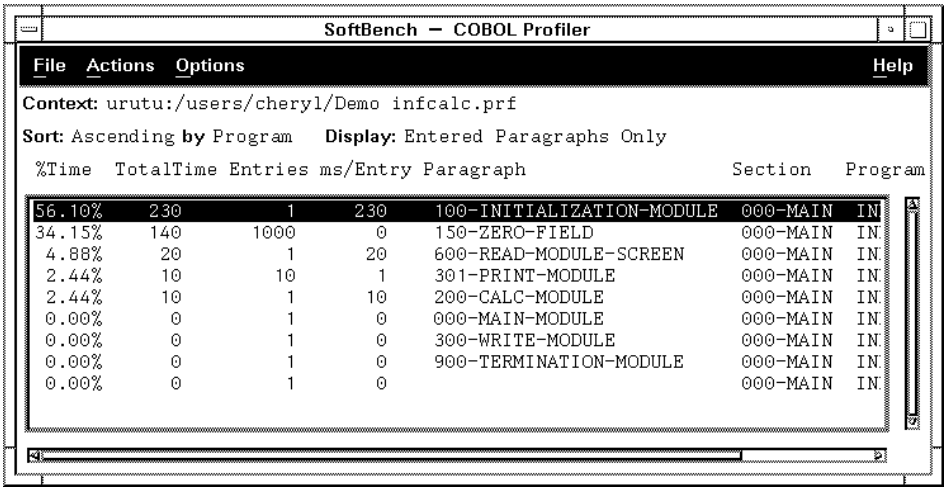

Fig. 5. The profile statistics for the calculation program.

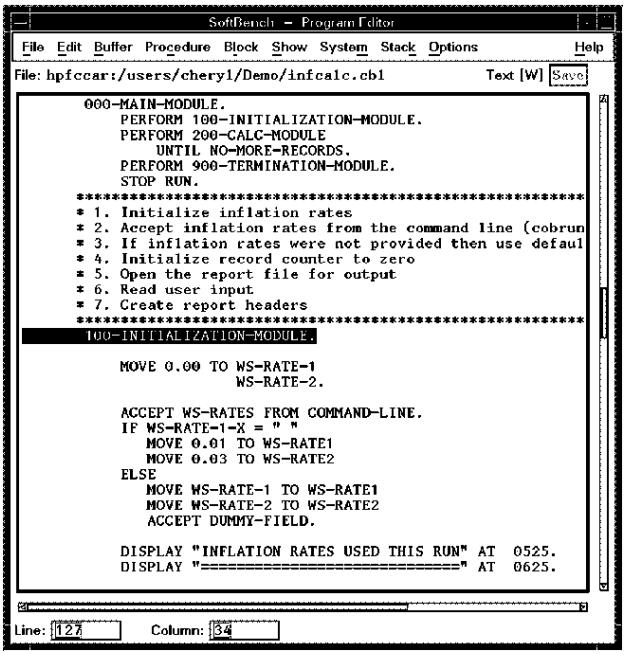

Fig. 7. The Softbench program editor showing the initialization module.

Program Structure. The SoftBench static analyzer allows the developer to save time in searching for program errors in unfamiliar COBOL code. From the SoftBench COBOL profiler, the developer double clicks on the line in Fig. 5 showing the initialization module. This action brings up the SoftBench program editor displaying the source code, starting at the beginning of the selected module (Fig. 7).

The developer needs to learn about the relationships between the many parts that make up the application without having to read the unfamiliar code. Using the mouse button, the developer drag-selects 000-MAIN-MODULE and chooses the Show: Definition( ) menu item to bring up the SoftBench static analyzer. Now, the developer can quickly display the program structure graphically using the Graph: Query Graph menu item (Fig. 8).

Existing Relationships. The static query graph shows that the initialization module performs the 150-ZERO-FIELD paragraph. Looking back at the SoftBench COBOL profiler in Fig. 5, the developer sees that this paragraph is called 1000 times!

Within a very short time, the developer knows where the program is spending time, has an understanding of the program structure, and is ready to debug at a specific paragraph. In other words, the developer has a good start toward solving the problem.

Program Animation. Programmers frequently need to understand a program's logic. By dynamically watching lines of source code execute, they can identify where the behavior of the program differs from what they intended.

Using the SoftBench COBOL animator, the developer selects the line of code that performs the paragraph in question, PERFORM 150-ZERO-FIELD UNTIL ZERO-COUNTER = 1000 (Fig. 9). Choosing Breakpoints: Zoom executes the program through the step just before the perform statement in question. While the animator is paused, the developer can view the code that is

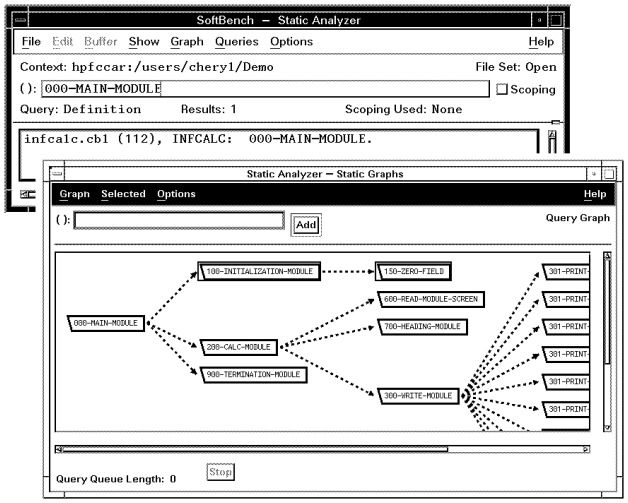

Fig. 8. The SoftBench static analyzer and the static graph showing the structure of the calculation program.

about to be executed. Looking at this paragraph reveals that the WS-ARRAY array elements are being set to zero.

The developer now selects the Animate button. The animate mode single steps through the program. As each line executes, watch windows show the name and value of all variables that change. The developer finds that the program executes this loop 1000 times, just as it was intended.

Textual Static Queries. The Show menu shown in Fig. 10 helps locate any place a specified identifier is referenced. From the SoftBench COBOL animator, the developer drag-selects the WS-ARRAY identifier and then chooses Show: References( ). There are only two references: the declaration and the perform statement.

A more experienced developer could have avoided the animator steps by editing the 150-ZERO-FIELD node from the static

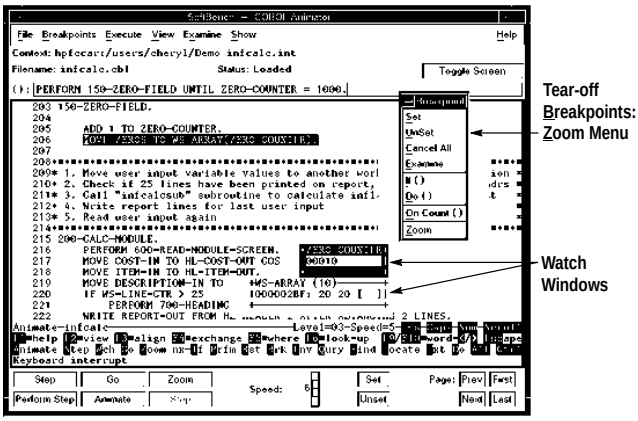

Fig. 9. The SoftBench COBOL animator showing a tear-off Breakpoints: Zoom menu and two examples of watch windows. A tear-off menu is a group of menu commands that have been "torn off" from their parent menu. The parent menu in this case is Breakpoints. A tear-off menu allows the user to keep on the workspace frequently used menu items even after the parent menu item is no longer selected. Fig. 3 shows case in which a menu could become a tear-off menu. By clicking on the dashed line, the menu items shown below the Actions command would become a tear-off menu.

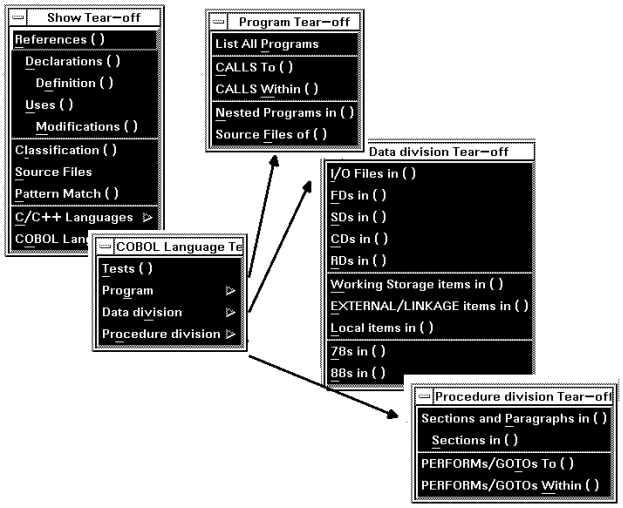

Fig. 10. The Show menu is used to help locate where a specified identifier is referenced in a program.

graph and checking the references for WS-ARRAY from the SoftBench program editor.

Source Code Modification. This defect fix is a simple one: delete the unnecessary perform loop. From the SoftBench COBOL animator, the developer chooses File: Edit and edits infcalc.cbl. Then from the SoftBench program editor, the developer searches for the ZERO-FIELD string, removes the unused code, and saves the changes to the file.

If the developer had needed to make major changes to the code, the SoftBench program builder would take away the pain of finding compile errors. After the developer saves and compiles the program, any errors found are displayed in the Build Output area. To browse and fix errors, the developer can select the First button or select any error in the list. The associated source file appears in the SoftBench program builder, with the text cursor located at the beginning of the line where the error was detected (Fig. 11).

Improved Application Performance. Before the code is returned to the version control system and production, it is important to confirm that a lower percentage of time is spent in the initialization module. The developer recompiles the program, with the Profile option still enabled, runs the program again and from the SoftBench COBOL profiler, selects the Entered Paragraphs Only display option. This shows that the 150-ZERO-FIELD paragraph is not entered during the application's execution (Fig. 12).

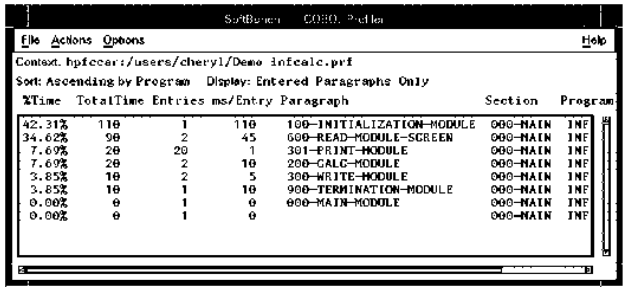

Fig. 12. Profile of the calculator program after it is fixed.

Mixed-Language Model

Development teams can take advantage of other technologies with COBOL SoftBench. For example, consider the developer who is writing a mixed-language application. One program module is written in COBOL, while another is a subroutine written in C. The COBOL module calls the C subroutine.

To illustrate the COBOL SoftBench model, consider the scenario in which the developer has set up a working directory using the SoftBench development manager and is using version control on the the two source files mixedtest.cbl and cpart.c. The program is not working as the developer expects. To help determine what is wrong with the program, the SoftBench COBOL animator and the SoftBench program debugger (for the C subroutine) are used.

Custom Run-Time System. With a mixed-language application, a custom run-time system is part of the application. From the SoftBench program builder, the developer creates a makefile, which detects mixed source code files. With this makefile, COBOL SoftBench will automatically create a custom run-time system. Next, the developer sets compile options to set up both programs for debugging and then builds the application.

To verify that the makefile is working as expected, the developer brings up the dependency graph browser (Fig. 13). This graph shows cpart.c is compiled to an object file and mixedtest.cbl is compiled to an intermediate target.

Mixed-Language Debugging. Since the COBOL program is the entry point in this example, the developer starts from the SoftBench COBOL animator, and the mixedtest.cbl source code is displayed (Fig. 14). The developer chooses File: Soft-debug Adopt to display the SoftBench program debugger window. After setting a breakpoint at the C procedure entry,

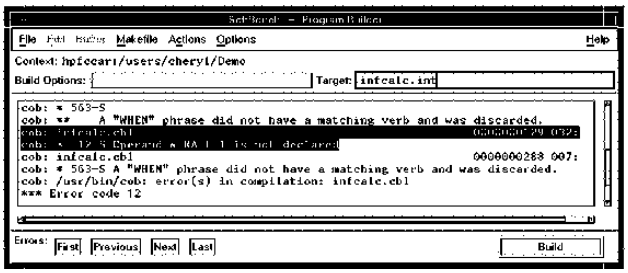

Fig. 11. The SoftBench program builder showing the line in the program where the error occurred.

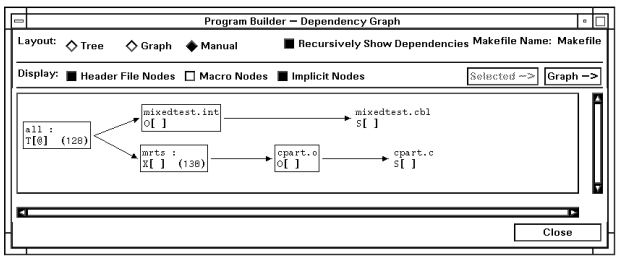

Fig. 13. Dependency graph to verify that the makefile is correctly identifying a mixed-language file.

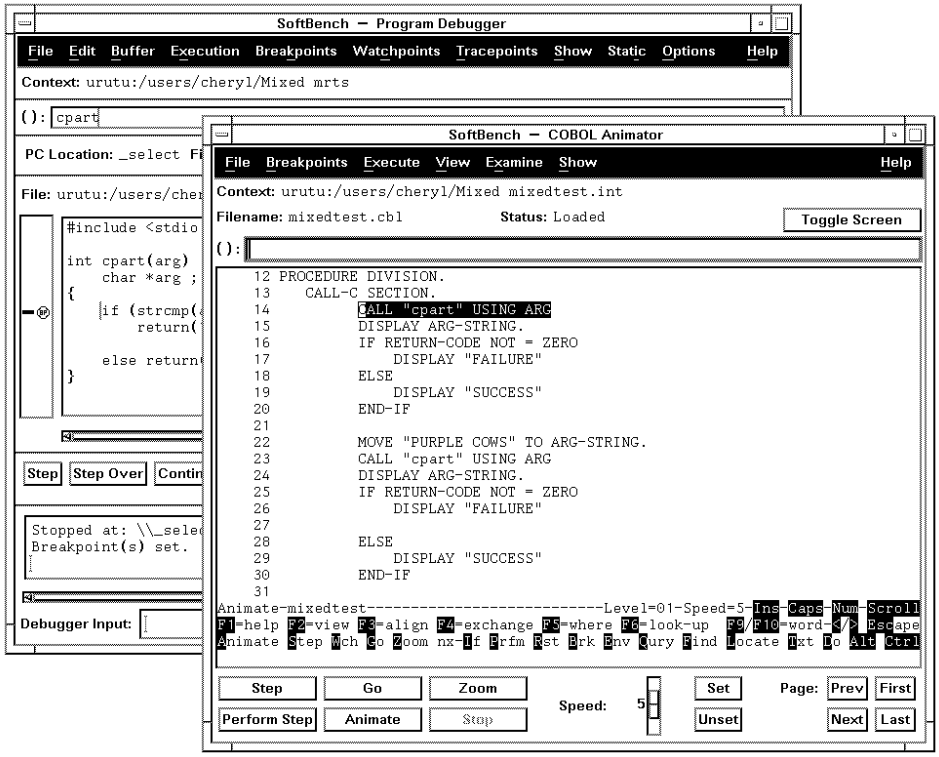

Fig. 14. The SoftBench COBOL animator and the SoftBench program debugger being used to debug a mixed-language program.

the developer selects the Continue button, which returns the control back to the SoftBench COBOL animator.

With both debuggers started, the developer is ready to debug the application using both the SoftBench COBOL animator and the SoftBench program debugger (collectively called debuggers).

Control transfers between the debuggers as the application executes the program module or the subroutine. The developer can apply all the features of either debugger, such as setting breakpoints, watching variables, or setting variable values.

COBOL and C++ Mixed Program. When a custom run-time system includes C++, the C++ language requires that the entry point be the C++ main function. The developer starts the SoftBench program debugger and the main.c source code is displayed. Choosing File: Animate Adopt displays the SoftBench COBOL animator window. The program begins execution at the entry point displayed in the SoftBench program debugger window, and again control passes between the debuggers, depending upon the application's behavior.

Conclusion

HP's integrated application development toolsets, COBOL SoftBench, COBOL/C SoftBench, and COBOL/C++ SoftBench help COBOL programming teams easily transition to open systems. In addition, these products help position the teams to transition to new application development technologies.

Acknowledgments

The COBOL SoftBench products involved many people. Special thanks go to our COBOL SoftBench product team managers Lee Huffman and Dan Magenheimer, our product marketing engineer Pat Hafford, my learning products team Margee Daggett and Tom Huibregtse, and our R&D team Lynnet Bannion, Paul Faust, Jay Geertsen, Robert Heckendorn, Mike Stabnow, Wade Satterfield, and lead engineer Alan Meyer. Additional gratitude to our partners at HP's General Systems Division and California Language Labs, and the Software Engineering Systems Division's core SoftBench team.

References

1. B.D. Fromme, "HP Encapsulator: Bridging the Generation Gap," *Hewlett-Packard Journal,* Vol. 41, no. 3, June 1990, pp. 59-68.
2. J. J. Courant, "SoftBench Message Connector: Customizing Software Development Tool Interactions," *Hewlett-Packard Journal,* Vol. 41, no. 3, June 1994, pp. 34-39.
3. C. Gerety, "A New Generation of Software Development Tools," *Hewlett-Packard Journal,* Vol. 41, no. 3, June 1990, pp. 48-58.