An Event-Based, Retargetable Debugger

Remote and event-based debugging capability, a sophisticated graphical user interface, and adaptability to different languages and target platforms are some of the features provided in this debugger.

by Arun K. Iyengar, Thaddeus S. Grzesik, Valerie J. Ho-Gibson, Tracy A. Hoover, and John R. Vasta

Software developers rely heavily on debuggers for both fixing bugs and analyzing programs. The information obtained by a debugger can be used to add new features and improve performance. Event-based debugging[1] is a powerful method for examining program behavior. In event-based debugging, the user instructs the debugger to respond to events that occur during program execution. The proper selection of events allows programs to be debugged at a higher level than would otherwise be possible.

This article describes the HP Distributed Debugging Environment (HP DDE).[2] HP DDE is distributed because it is capable of debugging programs executing on remote hosts. HP DDE has been ported and retargeted to several platforms and can be used to debug programs written in C, C++, FORTRAN, Pascal, and various assembly languages. The platforms that HP DDE can run on include the HP-UX* operating system, Domain/OS 68K, Domain/OS Prism, the SPARC implementation of Solaris, and the PA-RISC implementation of the OSF/1 operating system. HP DDE has a sophisticated graphical user interface that provides the user with point-and-click access to commands and program state. Finally, HP DDE has many features that support event-based debugging and debugging optimized code.

The modular architecture of HP DDE enhances its portability and has been a critical component in its success. HP DDE consists of a main debugger that communicates with several modules called *managers.* The main debugger contains support for generalized debugger functions, and the managers contain dependencies on specific languages, object code formats, target platforms, and user interfaces.

User-Visible Features

Event-Based Debugging in HP DDE

Almost all debuggers support a traditional debugging paradigm in which the user inserts breakpoints before critical points in the program and examines the state of the program after breakpoints are hit to find program errors. The disadvantage of this paradigm is that the user has to know where to insert breakpoints within the program. The user may also have to examine the state of the program at a number of breakpoints before obtaining the desired information.

In event-based debugging, the debugger does not return control to the user until an event defined by the user occurs. By choosing an appropriate event, the user can avoid stopping the program at points that are not critical. Event-based debugging allows the user to identify what is going on in a program without spending large amounts of time inserting breakpoints at crucial points.

Event-based debugging is achieved by monitoring the executing program at an interval or granularity level specified by the user. Whenever the debugger determines that a desired event has occurred, the debugger returns control to the user. A lower level of granularity, corresponding to frequent monitoring, allows the user to determine more accurately the location where a particular event occurs. However, a low granularity level can cause execution time to increase substantially. HP DDE provides several intervals for monitoring program execution, including every machine instruction, every source statement, and entry to or exit from each procedure. Monitoring can be restricted to specific procedures within a program. Typically, a user would use procedure-entry-and-exit-level granularity to narrow an event to a specific procedure. After the faulty procedure is located, the user can use source-statement-level or machine-instruction-level granularity within the procedure to determine the exact location of an event.

HP DDE contains many features for event-based debugging including conditional breakpoints, execution traces, data watchpoints, and event intercepts. Conditional breakpoints allow the user to halt execution at a given point in a program only if a set of conditions are met. Execution traces suspend program execution at intervals specified by the user. Data watchpoints allow the user to monitor a variable or memory range (program execution is suspended when a value corresponding to a watched variable or memory range changes). Event intercepts allow the user to regain control of the program when events such as program exceptions, shared library loading and unloading, thread creation and termination, and signals from the operating system occur.

The HP DDE command language allows the user to declare variables for use in a debugging session and contains loops, conditionals, and assignment statements to allow a wider range of event specification.

Examples

Suppose that the user desires to suspend execution every time a variable x becomes zero. This can be accomplished with the following HP DDE command:†

```
dde> watchpoint  x –do [if (x != 0) –then (go)]
```

The behavior when no monitoring interval is specified in the watchpoint command (as in this example) is to monitor the expression after every source statement.

As another example, suppose the user wishes to break upon entry to function foo only if argument x is nonnegative. This can be accomplished using a conditional breakpoint:

```
dde> breakpoint foo –do [if (x < 0) –then (go)]
```

It is possible for the user to examine data structures while a program is suspended. The following HP DDE commands will print all positive elements of a 20-element array A:

```
dde> declare int dde_var
dde> set dde_var=0
dde> while dde_var < 20 –loop [if A[dde_var] > 0 –then \
        (print A[dde_var]);\
        set dde_var = dde_var + 1]
```

The declare command allocates space for new variables that can be used in debugging sessions, and the backslash (\) indicates that an HP DDE command continues on the next line.

The following commands will suspend execution whenever the sum of the 20 elements of A is greater than 100:

```
dde> declare int i
dde> declare int sum
dde> watchpoint A –do \
      [set i=0; set sum = 0; \
      while i < 20 –loop \
         [set sum = sum + A[i]; set i = i + 1]; \
      if sum <= 100 –then (go)]
```

Sequences of commands such as these can be stored in a file and used as input to HP DDE. In addition, macro expansion allows a user to define a single command that is automatically expanded by the command interpreter into multiple commands.

As a final example, the following commands will execute a program and print the number of times a multithreaded application switches between different threads:

```
dde> declare int ts_count
dde> set ts_count = 0
dde> intercept thread_switch –do [set ts_count = ts_count + 1; go]
dde> go; print ts_count
```

The User Interface

Several user interfaces have been designed for HP DDE. The most sophisticated is a graphical user interface based on the X Window System and OSF/Motif.[3] It features multiple windows, context-sensitive pop-up menus, and online help. The user can customize menus, command buttons, and key bindings. A typical HP DDE debugging session with four windows is illustrated in Fig. 1. The graphical user interface can manipulate up to six different types of windows:

- Target program window. This is the window from which HP DDE is invoked. All data to and from the program being debugged is directed to this window. The main parameter to the debugger is the name of the program to be debugged.
- Transcript display window. This is the debugger's main window. It has four components. The command entry line component at the bottom of the window is for keyboard entry of debugger commands. All debugger commands can be entered from the keyboard in this area. Above the command entry line is the transcript area, which displays input to the debugger (including commands accessed from the mouse) and debugger output. The buttons along the left side of this window provide quick access to common debugger commands. For example, the user can single-step the program (execute a single source statement) by clicking on the button labeled Step. These buttons can be reconfigured by the user to represent other commands. The pull-down menus across the top of the transcript display window allow access to all debugger commands via the mouse.
- Source code display window. This window displays the source code for the program being debugged.
- Assembly display window. This window displays the disassembled machine instructions for the program being debugged.
- Traceback display window. This window displays the current dynamic call chain of the target program.
- Variable display window. This window displays the values of variables or memory ranges for which watchpoints have been set. When a watched value changes, HP DDE suspends execution, notifies the user of the change in value, and updates the variable display window with the new value.

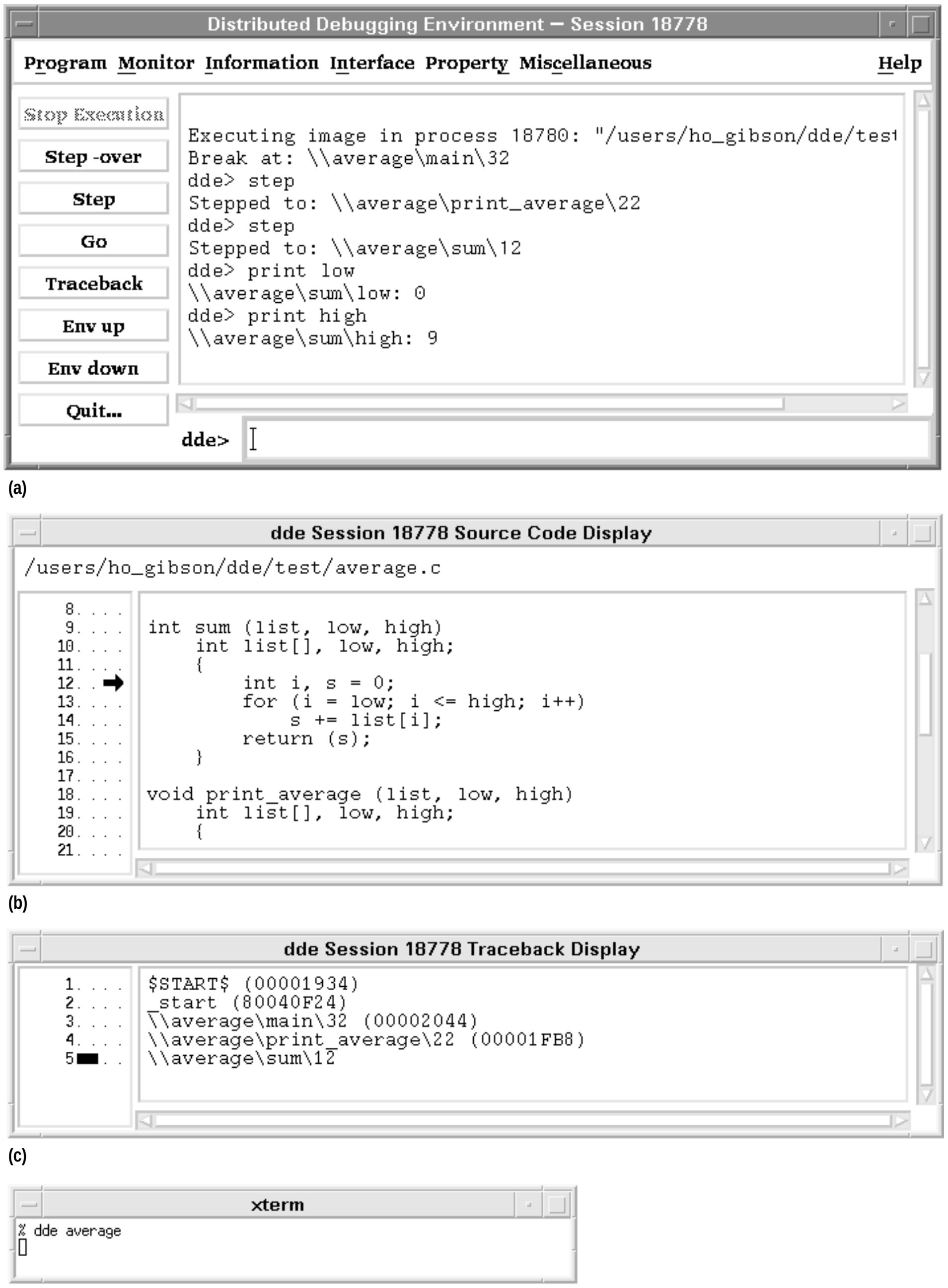
† HP DDE debugger commands can be entered in the command entry line area in the transcript display window described in the next section.

Context-sensitive pop-up menus allow the user to access common debugger commands with the mouse. Menus appear when the mouse is clicked. The menu that appears depends on the position of the mouse within a particular window at the time the mouse is clicked. For example, clicking on the mouse with the cursor positioned in the source code region brings up the menu displayed in Fig. 2. The cursor in Fig. 2 was positioned at the opening bracket of the expression list[i] at the time the mouse was clicked. The menu was thus seeded with the expression list[i]. If the cursor had been positioned over list, the menu would have been seeded with list instead of list[i]. The mouse can be positioned to print complex expressions in source programs without requiring the user to type in the expressions.

For even quicker access, double-clicking causes HP DDE to execute the first menu item instead of displaying the menu. The default pop-up menus contain actions that are commonly performed by users. However, pop-up menus can be easily reconfigured by users to contain actions that are common to a particular application.

Debugging Optimized Code

At a high level, optimization can be described as modifying instructions and memory references to improve program performance. Multiple source statements may map to the same machine instruction. In some cases source statements may not correspond to any machine instructions. Variables may be moved from memory into a register or from one register to another to eliminate costly memory references. Unless a debugger has some way of knowing how source statements and machine instructions have been rearranged and where a variable is located, it can be very difficult to


Distributed Debugging Environment — Session 18778
Program Monitor Information Interface Property Miscellaneous Help
Stop Execution
Step -over
Step
Go
Traceback
Env up
Env down
Quit...
dde>
(a)
dde Session 18778 Source Code Display
(b)
dde Session 18778 Traceback Display
(c)
xterm
(d)

```
Executing image in process 18780: "/users/ho_gibson/dde/test
Break at: \\average\main\32
dde> step
Stepped to: \\average\print_average\22
dde> step
Stepped to: \\average\sum\12
dde> print low
\\average\sum\low: 0
dde> print high
\\average\sum\high: 9
```

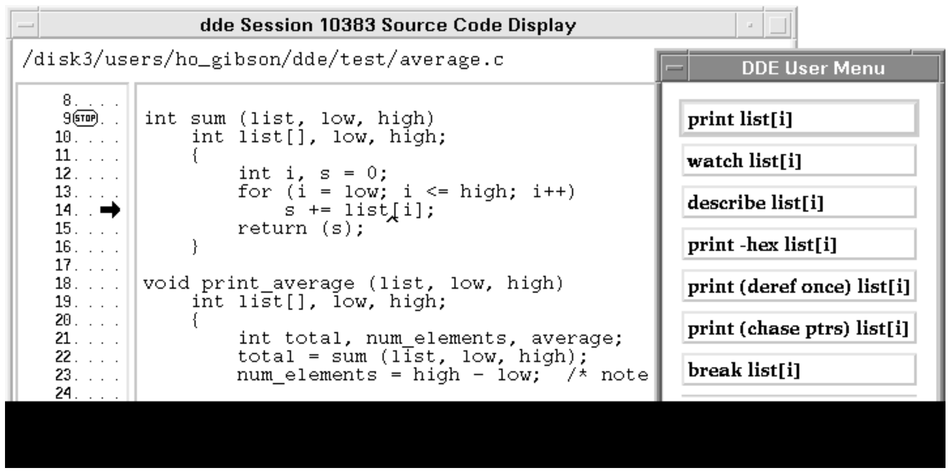
```
/users/ho_gibson/dde/test/average.c

 8. . . .
 9. . . .  int sum (list, low, high)
10. . . .      int list[], low, high;
11. . . .      {
12. . ➡        int i, s = 0;
13. . . .          for (i = low; i <= high; i++)
14. . . .              s += list[i];
15. . . .          return (s);
16. . . .      }
17. . . .
18. . . .  void print_average (list, low, high)
19. . . .      int list[], low, high;
20. . . .      {
21. . . .
```

```
1. . . .  $START$ (00001934)
2. . . .  _start (80040F24)
3. . . .  \\average\main\32 (00002044)
4. . . .  \\average\print_average\22 (00001FB8)
5■ . .    \\average\sum\12
```

```
% dde average
```

Fig. 1. Some of the windows involved in a typical HP DDE debugging session. (a) Transcript display window. (b) Source code display window. (c) Traceback display window. (d) Target program window.


dde Session 10383 Source Code Display
DDE User Menu
print list[i]
watch list[i]
describe list[i]
print -hex list[i]
print (deref once) list[i]
print (chase ptrs) list[i]
break list[i]

```
/disk3/users/ho_gibson/dde/test/average.c

 8. . . .
 9(STOP). .  int sum (list, low, high)
10. . . .      int list[], low, high;
11. . . .      {
12. . . .          int i, s = 0;
13. . . .          for (i = low; i <= high; i++)
14. . ➡              s += list[i];
15. . . .          return (s);
16. . . .      }
17. . . .
18. . . .  void print_average (list, low, high)
19. . . .      int list[], low, high;
20. . . .      {
21. . . .          int total, num_elements, average;
22. . . .          total = sum (list, low, high);
23. . . .          num_elements = high - low;  /* note
24. . . .
```

Fig. 2. A context-sensitive pop-up menu. This menu was obtained by positioning the mouse cursor (denoted by the caret on line 14) in the source code display window and clicking on a mouse button.