利用自動錯誤處理於快速發展應用程

Using Automatic Error Handling in Rapid Application Development

吳培基

國立澎湖海事管理專科學校資訊工程科

澎湖縣880馬公市六合路300號

Email: pcwu@computer.or

摘　要

今日許多資料處理應用都是設計在主從式架構的環境執行。前端程式主要是一圖形使用者介面；主機端的程式則主要是資料庫及相關的程序。利用快速發展應用程式的技術，前端程式可以很快的製作出來，使用者則可以儘早參與發展的工作。然而，初期發行的雛形版本通常含有相當多的臭蟲，使得雛形難以使用。本文提出利用自動錯誤處理於快速發展應用程式的方法及經驗。本文的自動錯誤處理分為二個步驟：1)報告錯誤；2)繼續執行。我們的實作採用微軟的Visual Basic以及其錯誤處理架構。本文也報告我們初期使用的一些經驗。

關鍵字：資料處理、主從式架構、雛形、資料庫、錯誤處理。

Abstract

Many recent data processing applications have been designed for client-server computing environments. The client program is mainly a graphical user interface; and the server program is usually a database with related procedures. By applying rapid application development (RAD) technique, the client side of data processing applications can be crafted quickly. Potential users can then involve in the development process as soon as possible. However, early releases of prototypes tend t contain bugs that make the prototypes difficult to use. This paper presents our experience in using automatic error handling for RAD. We define automatic error handling as two steps: (1) report errors and (2) continue execution. Our implementation uses Microsoft Visual Basic and its error handling constructs. Preliminary experience on the implementation is also reported.

Keywords: data processing, client-server, prototypes, databases, error handling.

1. INTRODUCTION

Many recent data processing applications have been designed for client-server computing environments. The client side consists of a number o graphical workstations; the server side is usually a powerful data processing engine. The client program is mainly a graphical user interface; and the server program is usually a data store with related procedures. This allocation of tasks simplifies the development of client-server data processing applications for the following reasons: 1) Most servers now support Structured Query Language (SQL) [2], which provides a uniform way to process data stored in relational tables. 2) The client side is usually well decomposed into the following tasks: user interaction, query composition, query results display, report generation, etc. Many vendors provide an environment to integrat a set of tools for these tasks. Using such an environment needs to deal with a set of tools and practices and is so called rapid application development (RAD) [4, p.516].

By applying RAD technique, the client side of data processing applications can be crafted quickly Potential users can then actively involve in the development process when the first prototype is available. This can help to find what users really need and what they dislike in the early stage of a project. However, early releases of prototypes tend to contain bugs that make prototypes difficult to use. For example input forms or menus may not be successfully invoked due to unexpected errors, such as null values, improper input data, memory leaks, disconnected network lines, etc. Sometimes, a prototype hangs with a message that says nothing.

Reporting these errors to developers in details is not users' duties and interests. This occasion is even worse when the development team has no dedicated software testing professionals to help to find errors.

Recent RAD environments provide not much help to this problem. A transaction failure in a database will be recorded in the database log; however, an error in a client program is usually ignored when the erroneous program terminates.

Another approach to this problem is by using in-depth analysis tools to detect errors in a program. There are many commercial tools (especially for C/C++ [3, 1]) that can detect errors such as memor leaks and uninitialized va iables. These tools are helpful to improve the quality of a program; however, even a carefully analyzed program may still caus unexpected run-time errors. In addition, these analysis tools are usually language-specific and may b unavailable to the programming language of the RAD environment.

This paper presents our experience in usin automatic error handling for RAD. We define automatic error handling as two steps: (1) report errors and (2) continue execution. Our design is based on programming languages with structured error handling, and the implementation is on Microsoft Visual Basic. The implementation has been used for months by the systems currently developed in ROC Airforce. Our preliminary experience shows that this technique is useful to identify bugs in the prototypes of the systems.

2. Requirements of Automatic Error Handling

In a client-server application, simply promptin a message like "a general error occurred" or "general protection fault" is not good enough, because users may first bypass this message and then be unable t describe their operations in details. To be specific, w define automatic error handling as two steps: (1) report errors and (2) continue execution.

The best place to record error messages is in the server side. Other related program information (call stacks, values of variables, etc.) can also be recorded t help developers understand what happened then. For data processing applications, the SQL statement recently sending to the server may be interestin too. Recording who was using the system is also useful if developers can contact that user for additional information.

Continuous execution is required to keep users patient when they are using early releases of prototypes. Continuous execution should be guaranteed any time except that fatal errors occur in operating systems or computer hardware. One way to recover a program from an erroneous state to a safe state is by structured error handling: errors that cannot be handled in the local context are propagated to the outer (larger) program structure until a safe state is found.

Most recent windo -like graphical user interfaces are developed using the event -driven programming technique. When the processing of an event causes an error, it is usually safe to rollback all database transactions and to terminate the processing directly. Because it is costly to undone all of the processing in the client side, we only require that the display screen remains in a normal state for user input. This can usually be achieved by simply setting the mouse pointer to a normal state.

For efficiency and simplicity, som programming languages (e.g., the C language [3]) do not provide structured error handling. Developin automatic error handling for these l anguages is thus more difficult than that with error handling constructs. Structured error handling does take space and time overhead: every statement and expression needs to b checked for potential errors. However, from the viewpoint of composition of s ftware components, such overhead can usually be ignored. Data processing applications developed by RAD technique usuall reuse many (de facto) standard software components. A software component such as a list box may contain thousands lines of codes. The overhead in structured error handling across component boundaries is thus very small in proportional to the size of components.

3. Design and Implementation

Our design is based on programming languages with structured error handling and cl ient-server computing environments. The implementation is on Microsoft Visual Basic 4.0 [5] (16-bit version), which contains a rich set of error handling constructs such as "on error goto label" and "raise error". Ou implementation consists of three parts: 1) a general error handler subroutine called `GENERAL_ERROR`, 2) a code translator called `lineno`, and 3) an error log viewer.

3.1 The general error handler subroutine:

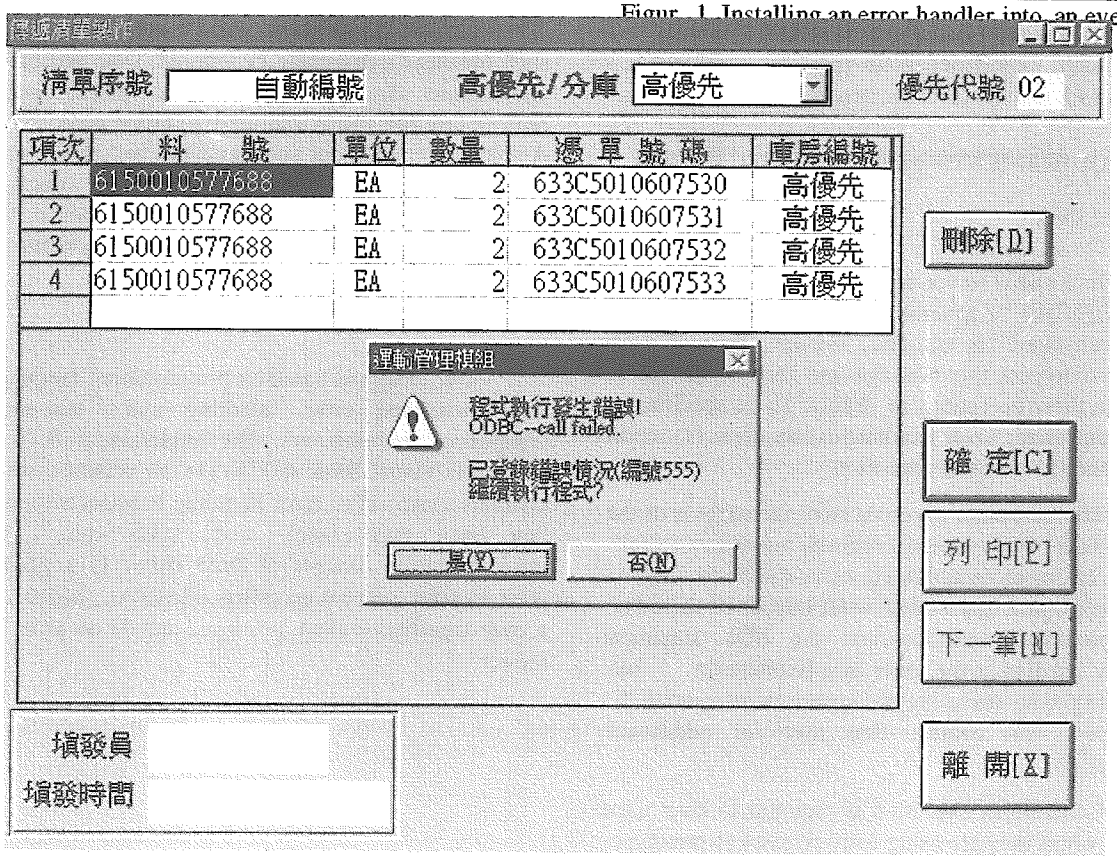
GENERAL_ERROR

When a run-time error occurred, subroutine `GENERAL_ERROR` first roll backs all database transactions and resets the mouse pointer. Subroutine `GENERAL_ERROR` then records the following information in the `ERROR_LOG` table: the user name, the subsystem name, the error number, the error message, the active form, the active control, the call stack, and the last SQL statement. The first two can be obtained from the program information. The rest of data except the call stack and the last SQL statement are available from Visual Basic's built-in functions and objects: `Err` (error number), `Erl` (error label), `Errors` (a collection of database access errors), and `Screen` (the screen display). Note that all database access errors are categorized as "ODBC—call failed", and the detailed information can be obtained from object `Errors`.

The last SQL statement is obtained from the `xDatabase` object used in the application program. We assume that an application access s only one data source, an abstract mechanism that can be mapped int several remote physical databases. The `xDatabase` extends Visual Basic's `Database` object with the function that stores the last SQL statement sending to the data source. The `xDatabase` has the same interface as `Database` does, so programs that originally access `Database` objects can remain unchanged.

```
Private Sub mskDate_LostFocus (Inde
As Integer)
  On Error GoTo GENERAL_HANDLE
  If Index = 1 And date_changed Then
    DoEvents
    date_changed = False
    Call query
  End I
  Exit Sub
  GENERAL_HANDLER: GENERAL_ERROR
"mskDate_LostFocus", False
End Sub
```

Figur 1. Installing an error handler into an event



Figur 2. A run-time error captured by the automatic error handling routine.

The call stack is a sequence of function names and line numbers indicating the execution path of a program. To collect the call stack, GENERAL_ERROR raises a special error to be handled by the outer handler, which calls GENERAL_ERROR again. This iteration collects each function name and line number of the call stack step by step. The iteration terminates until an outermost function is reached. There are two kinds of outermost functions in Visual Basic: 1) the subroutine Main, and 2) the event handlers of forms and controls. All forms and controls are independent in handling input events, so every event handler needs an error handler. In Visual Basic 4.0, there is no simple way to write just one general error handler for all of the forms in an application. Figure 1 shows how to install on error handler in the event handler of th LostFocus event of mskDate control. GENERAL_ERROR takes two arguments: sfunction, the name of function/subroutine and trace, whether or not to g to the outer context. Th trace flag is set False here, because mskDate_LostFocus is an outermost function and does not need to trace the outer context.

Figure 2 shows a run-time error captured b GENERAL_ERROR. GENERAL_ERROR pops up a message box with a brief description of the error ("ODBC—call failed."), the log ID (555) in the ERROR_LOG table, and two options: "continue" (Yes) and "terminate" (No). If the user choose "continue" (by default), the program returns to the normal input state. Otherwise, the program terminates.

3.2 The code translator lineno

The lineno translator processes the Visual Basic source codes and installs an error handler in every function and subroutine. The translator als inserts a line number for each statement. This translation process adds about 10% space overhead t the compiled execution files. Figure 3 shows the output of function del_nl processed by the translator. The translator adds all the statements for automatic error handling. The arguments passed to GENERAL_ERROR are sfunction="del_nl" and trace=True, because del_nl is not an outermost subroutine. Not that the line number for the first "Case" statement (No. 6) is skipped due to the syntax restriction in Microsoft Visual Basic

The translator is handwritten in the C languag and contains about 500 lines of code. The translation is not a complete parsing of Visual Basic codes. It matches only some Visual Basic's key words: begin, end, private, public, sub, function, on, error, goto, select, and case. Most statements are simply expanded with line numbers. When a input statement contains a line number, the translator generates a warning message. Key words "begin" and "end" of a function/subroutine are matched to install an error handler for each function/subroutine. Because the statement "on error goto 0" will disable the current error handler, it is replaced by "on error goto GENERAL_HANDLER". To avoid incorrect self-recursion, the statements in the subroutine GENERAL_ERROR itself are skipped during processing.

3.3 The error log viewer

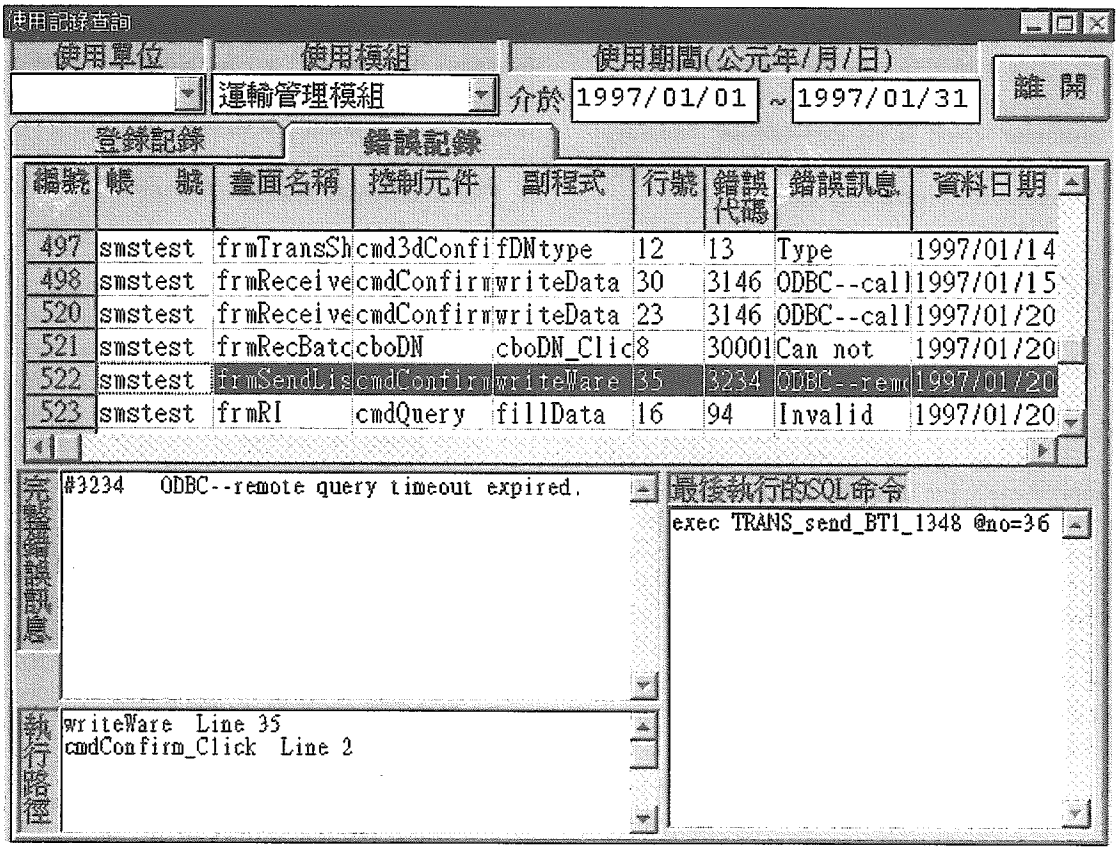
Figure 4 shows an error logged in the server. The top is a grid of error list. The columns of the grid (shown in Chinese) are as follows: the log ID, the user name, the form name, the control name, the subroutine name, the line number, the error cod , the error message, and the log date. Data access errors and the call stack are shown in bottom-left; the last SQ statement is shown in bottom-right. Clicking a row of the grid makes the row highlighted and the related information displayed in the bottom of the form. Developers can view the error log easily and obtain the detailed error report in few seconds.

4. PRELIMINARY EXPERIENCE

The implementation presented has been used for months by the systems currently developed in the author's site. We have released to users the prototypes with automatic error handling function and hav collected hundreds of run -time errors. Table I summarizes the errors collected from an installed prototype. Some errors are unexpected in laborator setting but occur very often in user's site. For example, many users are connected to the server with a lo speed modem, and database query time-out (the error "ODBC—remote query timeout expired.") occurs when the network or the server is busy. There are als errors due to incorrect versions of client programs, database schema changes (e.g., some of the "ODBC—call failed" errors), and improper setting of client sid environments (e.g., the error "Can't load (or register) custom control").

Our preliminary experience shows that this technique is useful to identify bugs in the prototypes of the systems. Developers can quickly identify the fault part of the system. Developers who are responsible fo

the bug can then take the detailed bug report from thei workstation and make the correction immediately. Communication time between users and developers can be greatly reduced. In most cases, there is also no need for users to write down bug reports in paper. We also find that many users do not response to developers when they encounter bugs in the system, although they were recommended to do so. This fact also indicates the need of automatic error handling mechanism in projects that emphasize early involvement of users.



Figur 3. The error log vie er.

```
Function del_nl(s As String) As Strin
On Error GoTo GENERAL_HANDLER
  1   'remove newlines (CarriageReturn & LineFeed)
  2   Dim i%, c$, tmp$
  3   For i = 1 To Len(s)
  4     c = Mid(s, i, 1)
  5     Select Case c
          Case Chr(13), Chr(10): ' removed
  7       Case Else:  tmp = tmp & c
  8     End Select
  9   Next i
 10   del_nl = tmp
Exit Function
GENERAL_HANDLER: GENERAL_ERROR "del_nl", Tru
End Functio
```

Figur 4. The output of a function processed `lineno` translator.

5. Conclusions and Future Work

This paper has presented our experience in using and developing automatic error handling for RAD. Our implementation has been used for months by the systems currently developed in the author's site. We have collected hundreds of run-time errors. Some errors are unexpected in laboratory setting but occu very often in user's site. Our preliminary experienc shows that this technique is useful to identify bugs in the prototypes of the systems.

The translator used in our implementation is a separate program to pre-process source codes. This pre-processing can be integrated into RAD environments such as Visual Basic's integrated development environment.

Acknowledgments

The author would like to thank his colleagues in ROC Air Force for their cooperation in experiment of this technique in several systems.

References

[1] Ellis, M.A., and Stroustrup, B., *The Annotated C++ Reference Manual*, Addison-Wesley, 1990.

[2] Elmasri, R., and Navathe, S. B., *Fundamentals of Database Systems*, The Benjamin/Cummings Publishing Company, 1989.

[3] Kernighan, B. W., and Ritche, D. M., *The C Programming Language*, 2nd Edition, Prentic Hall, 1988.

[4] McConnell, S., *Rapid Development: Taming Wild Software Schedules*, Microsoft Press, 1996.

[5] Microsoft, Microsoft Visual Basic Enterprise Edition, Version 4.0, 1995.

Table I. Errors collected from an installed prototype.

Error Description	Count
ODBC--call failed.	48
Can't load (or register) custom control	19
Subscript out of rang	16
No current record.	15
Type mismatch	14
Invalid procedure call	13
ODBC—remote query timeout expired.	9
Object variable or With block variable not set	6
Fil not found	4
Item not found in this collection.	4
Object doesn't support this property o method	4
Overflow	3
Path/File access error	2
Bad DLL calling convention	2
Object is invalid or not Set.	2
Out of memory	1
Path not found	1
Invalid use of Null	1
Invalid Column Value	1
Unable to activate object	1
Can not remove last non-fixed row	1
Can not do an AddItem in a fixed ro	1
Application-defined or object-defined error	1
'Height' property cannot be set within a page	1
Problem getting printer information from the system	1