Perl, the Enterprise Panacea?

Monitoring SAP R/3 with Perl

David Moring , August 1999

The Perl Conference 3.0

Introduction

This paper outlines a solution for monitoring SAP R/3 performance using Perl and Perl/Tk. This solution provides a primitive monitoring application which demonstrates how performance data extracted from SAP R/3 can be graphically displayed. In addition, the monitoring ABAP function within SAP R/3 can be invoked from within a Perl script and the data can subsequently be analyzed using Perl.

The Problem

As an enterprise system, SAP R/3 provides several different ways to tune the system, the principal means is the internal tool, known as the Computing Center Management System (or CCMS for short). While CCMS provides a rich set of performance information for the SAP R/3 system itself, the information is limited to the SAP R/3 view of the world. For example, the information is collected by a internal program (known as the COLLECTOR_FOR_PERFORMANCE-MONITOR) that gathers metrics of the system and stores it in a database table (principally the table MONI). This information is from the SAP R/3 system itself. Since the SAP R/3 system itself is just one component of an enterprise business process, other useful information, regarding other systems, is not available.

Since the SAP R/3 perspective of performance management is introverted, there is often the need to understand how SAP R/3 is performing in relation to other systems. This can be viewed as managing SAP R/3 from the business process perspective, in which SAP R/3 participates, but is not the only business system. Consider a petroleum solution that is in use today as an example. Petroleum is pumped out of the ground, the ground above (including the mineral rights) is owned by several different people. The amount of the oil that lies under the ground is apportioned out percentage wise using formulas. The result is that a single well can owe royalties to hundreds of interests at a different percentage for each. SAP R/3 could possibly solve this problem, with lots of custom coding and the like, or external systems, which do this well, can be leveraged.

To manage this type of business process, metrics from each of the systems must be collected. This requires that the performance information within SAP R/3 must be externalized in an easy to use manner. That is the focus of the rest of this paper.

The Solution

The solution is to run an internal SAP R/3 performance report, and then to export that report as a flat file. The flat file is then parsed via Perl to extract the needed information. The information is then analyzed and displayed in a Tk application. Graphically, the process can be modeled as Figure 1: Program Flow.

Invoking the Performance Report

The first step in the process is to invoke an internal SAP R/3 report to export performance data. To do this, the report RSSTAT10 was selected. This report, given the proper variants, will produce a highly detailed report for each of the system task types. For this paper the TOTAL task type was used, as it is the aggregate of the SAP R/3 system.

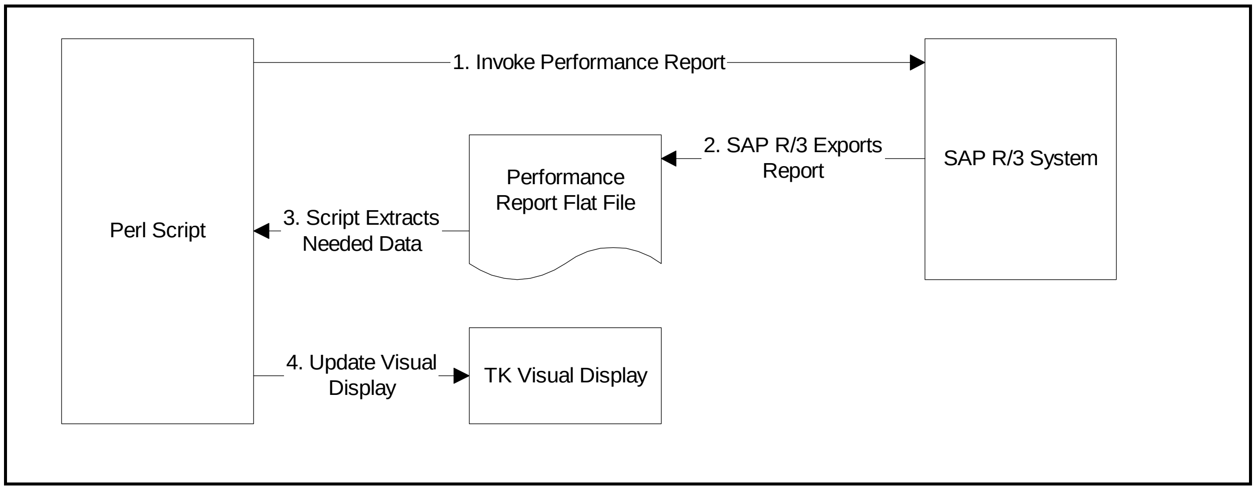
1. Invoke Performance Report
2. SAP R/3 Exports Report
SAP R/3 System
Performance Report Flat File
3. Script Extracts Needed Data
Perl Script
4. Update Visual Display
TK Visual Display

Figure 1: Program Flow

To invoke this report a brief SAP R/3 ABAP was written.[1] This report was then designated within SAP R/3 as externally callable via the external SAP R/3 RFC interface. The report can then be triggered from an external call. This report could also be scheduled using the internal SAP R/3 scheduling system to execute at given intervals.

The SAP R/3 ABAP

Here is the ABAP report in its entirety:

```
FUNCTION Z_DGMTP3_EXPORT.
*"----------------------------------------
--------------------------------
*"*"Local interface:
*"       EXPORTING
*"----------------------------------------
--------------------------------

  TABLES: RFCRECORD.
  DATA: BEGIN OF MITAB OCCURS 0,
   MYRFCSIZE LIKE RFCRECORD-RFCSIZE,
   MYRFCRECORD LIKE
RFCRECORD-RFCRECORD,
 END OF MITAB.

  DATA FNAME(60) VALUE 'dgm_perf_data'.

  DATA OUTTAB(150) OCCURS 0 WITH HEADER
LINE.

* Begin by running the systems manage-
ment report RSSTAT10 and capture
* results to memory, here we have hard
coded the system information
* TODO: change each of these required
data to importing

  SUBMIT RSSTAT10
      WITH SSYSTEM = 'asystem'
      WITH SPERIOD = 'D'
      WITH SDATE = SY-DATUM
      WITH TASKTYPE = '*'
      WITH REPOFLAG = 'X'
      EXPORTING LIST TO MEMORY
      AND RETURN.

* TODO:  Catch any exceptions here

*  Get the results from memory, for
later processing
*  use the MITAB internal table to hold
them

  CALL FUNCTION 'LIST_FROM_MEMORY'
       TABLES
            LISTOBJECT = MITAB
       EXCEPTIONS
            NOT_FOUND  = 1.

*  TODO:  Catch exceptions

*  Now change the gibberish ABAP report
native to something useful (ASCII)
*  Put the results into the OUTTAB --
Just in case we want to do something
*  later with them

  CALL FUNCTION 'LIST_TO_ASCI'
*    EXPORTING
*         LIST_INDEX = -1
     TABLES
            LISTASCI = OUTTAB
            LISTOBJECT= MITAB
     EXCEPTIONS
            EMPTY_LIST         = 1
            LIST_INDEX_INVALID = 2
            OTHERS             = 3.

*  TODO: Catch exceptions
  OPEN DATASET FNAME FOR OUTPUT IN TEXT
MODE.
  LOOP AT OUTTAB.
      TRANSFER OUTTAB TO FNAME.
  ENDLOOP.
ENDFUNCTION.
```

Use Perl to Fire the Report

Now that the ABAP function is written, and has been designated as externally RFC callable, it can be applied.

A stub function is required to call the SAP R/3 RFC, this can be generated using internal SAP R/3 tools to create the C source code. The resulting executable allows the function to be called from the command line or via Perl.[2]

This allows the report to be called through the SAP R/3 RFC interface. Again, this is not rocket science and the best way to do this (lo and behold) is to use a line of Perl script (very short), such as follows:

```
system "./wr3rfc -d DEM -c 000 -u
sapuser -p password -h asystem -s 00
ConfigFile";
```

The wr3rfc program requires a “side info file” that follows (for completeness):

```
COMMAND OPTIONS:
MODULE = Z_DGMTP3_EXPORT
```

[1] If this were a production ABAP program, there would be a great deal more commenting and complete exception processing routines would be included.

[2] To save some time, I used a SAP R/3 RFC wrapper program “wr3rfc.exe” that is shipped with the Tivoli Module for SAP R/3 (Tivoli and IBM are trademarks of IBM). Consult the SAP R/3 documentation on creating your own, there is an open source way.

Use Perl to Parse and Analyze Flat File

Now that the data is in a text file, the power of Perl's regex can be brought to bear. In this case, the response time of the SAP R/3 system will be extracted. The Perl program will parse through the program and pull the response time and assign it to a variable. The same will be done with the average database request time, the average CPU time, and the wait time. These metrics have a relationship. The response time is the sum of the wait, CPU, and database time (see Figure 2: SAP R/3 Performance Metrics).

Average Response Time		
Average Wait Time	Average CPU Time	Average Database Request Time

Figure 2: SAP R/3 Performance Metrics

The average response time should be about one second in a well-sized, tuned SAP R/3 system. The CPU time and database time should not make up more than 40% of the average response time. Given these metrics, and the performance guidelines, now a basic SAP R/3 performance monitor can be built.

Use Perl to Analyze the Data

The average database request time can now the correlated with the response time to provide a percentage. This can show how close the database request time is exceeding the limit. Likewise, the CPU time can be correlated with the response time to determine how close the CPU time is to exceeding the limit. The overall response time can also be compared to 1 second to gage performance. This can also be done by subroutines within the script, as is demonstrated in the next section.

The Perl/Tk Applet

To make this interface even more interesting a quick Tk program was pulled together to provide information in one glance.

Program Description

The Perl program used in this project performs the following functions using a graphical interface:

1. Calls the ABAP Performance function via the SAP R/3 RFC interface.
2. Parses the resulting flat file for needed data.
3. Displays/Updates data in a graphical interface.
4. Allows a refresh interval to be set.
5. Help is included.
6. Open design for easy customization and extensibility.

Program Features

This program was written to be quickly expandable and adaptable, this was done using the following:

1. The data is stored in a anonymous array of hashes. The entire array is parsed by the display and update routines, so additional data types can be added by just adding another hash.
2. The hash array for each data type that is extracted contains the regex allowing quick customization to extract other data.
3. The hash array for each data type contains the file name, allowing the data to be pulled from several different flat files, allows monitoring of other SAP R/3 reports or other systems' flat files.

The program is broken into subroutines to allow portability and reuse.[3]

A Walk Through the Program

The program begins with the creating of the anonymous list of hashes. The main window is then created, and the initial title is assigned. A status frame is created and the balloon help is initiated. The file menu is then created. A main frame is created.

This completes the initial setup. The hash list is parsed and the preliminary data is entered. The control is then passed off to the routine that updates, processes and fills the data. The main program ends in the MainLoop call, as all Tk apps are prone to do.

This fill frame routine calls the routine that retrieves the data. The routine that processes data is then called. This data is then used to update the Tk display.

The routine that retrieves data parses the data hash array. For each hash, the file is opened in slurp mode, the regex is executed, and the retrieved data is stored.

[3] This program stole great amounts of code from an Internet stock tracking program that I wrote, as such, the use and structure of the previous program allowed this program to be built in about 10 hours of time. Fast for coding.

The routine that processes the data is bit more messy. It is not likely to be as exportable as other parts, hence its encapsulation (well almost, the hash array is program wide scope, as does some other chunks of data) of the processing logic. This routine compares the data and assigns the appropriate alert level and balloon message.

Usage

This program was written to be easy to use (did not have time to write a help file). The menu allows the information to be retrieved once, or at regular intervals selected. The display contains pop-ups that provide extra information and a status bar that displays hints. The time of the last run is also displayed on the status bar.

The Conclusion and The Irony

This paper has demonstrated how to extract performance data out of SAP R/3 and to convert data to meaningful knowledge regarding SAP R/3 performance. It is admitted that this solution does not provide any real enhanced information that cannot be obtained internally using the SAP R/3 tool CCMS. That was not the intention.

Instead, it has been demonstrated how to externalize SAP R/3 operational information to the rest of the world using Perl. This now allows the SAP R/3 system's performance information to be leverage in enterprise business process instrumentation. Which, in turn, allows the business process owner to focus on the entire business process and not just the internal SAP R/3 workings.

Perhaps it is ironic that a multi-million dollar, business critical system can be monitored by a "free" open source solution. Contra-wise, maybe it is a statement of perl's elegance and power, the brilliance of the vision and the sweat equity of those that have contributed their time and effort. Either way, it works.

Comments and questions are welcome, please address, via the e-mail, to david@tivoli.com.

All trademarks and/or trade names listed are owned by the respective companies. This paper is an independent work, and is presented as such.