Successfully Developing On-Line RPG/3000 Applications

Duane Schulz
Hewlett-Packard Company
Wilsonville, Oregon

INTRODUCTION

Why do people laugh when I talk about interactive RPG/3000 applications? This paper will focus on the unique nature of RPG as an on-line language on the HP3000 computer system, identify the difficulties to be anticipated in learning how to best use the HP3000 with RPG as the primary language, and attempt to outline a clear path to success in this area, with attendant suppression of the abovementioned laughter. Hopefully, RPG users who review this paper will be able to identify and alleviate any current problems they are experiencing as interactive RPG users, and new users will be able to make the transition to the HP3000 smoothly.

To accomplish this, it will first be necessary to profile the probable assumptions of a new RPG/3000 user, outline the nature of the HP3000 computer, and identify any dissimilarities between the two. Once this is done, a series of steps designed to reconcile differences between RPG users' understanding of computers and the nature of the HP3000 will be presented. The reconciliation of these differences is the only true obstacle to success as an RPG/3000 user, though this basic fact is usually ignored while technical symptoms of the problem are addressed instead. the goal of the presentation is to help RPG users to finally come to grips with the struggle which is the inevitable result of conversion to the HP3000 while still relying on ideas and concepts which do not apply to the HP3000.

A USER PROFILE

In order to avoid stereotyping, I will describe my personal background at the time I first selected the HP3000 as a replacement to an IBM GSD computer. This background is very common, given the vast number of IBM System/3 and /34 systems currently installed. Also note that I'll assume that RPG/3000 users received their initial programming experience on IBM equipment; they set the de facto industry standard. The following statements summarize the computing concepts I employed at the time of my first conversion (please don't assume that these matched my philosophy about computing or my experience in actual implementation. . .):

- Transactions enter the computer (from the data processing department) in batches which must be 100% correct before the resulting data (usually a report) was returned to the user, again via data processing personnel.
- Each system was controlled by one user, and was 90% unrelated to other systems on the computer. Sharing common data was rare.
- Most technical solutions and learning was vendor-provided. User groups served other (social, system back-up) purposes.
- Interactive updates were best performed on dummy files, with the real updates performed during off-hours in a batch fashion.
- The system was best used for serial I/O.
- Multiple tasks were accomplished through fixed memory partitioning or special control programs (ie. CCP, etc.).
- Databases involve high overhead and long learning processes.
- Knowledge of operating system internals is not related to successful application development.
- Efficient programs were related to avoidance of high-overhead calculations and program logic.
- Data structures, screen handlers, and internals were best learned through RPG interfaces to such facilities. Knowledge of the specific facility on its own was not necessary (or, in some cases, possible).

HP3000: BASIC ASSUMPTIONS

Thought we all understand or have learned at least some of the following assumptions, it is important to note the nature of the HP3000 as compared with the mind set described above. Again, this is a list of statements or descriptions highlighting the important concepts behind the HP3000. These things must be learned before we can develop applications for the 3000 and expect them to be stable or efficient.

- The HP3000 is designed to be an interactive, user-driven computer, with transactions occurring randomly, usually with little involvement on the part of the data processing department.
- Most application systems will be (logically if not physically) related to one another.
- Learning and technical solutions are provided by

the vendor, a number of user groups, appropriate third-party assistance, and especially through self-teaching and exploration on the user's part.

- The system is designed for for interactive use. Serial I/O is one of the least efficient I/O techniques.
- The operating system includes a memory manager and dispatcher which allow dynamic memory sharing between users.
- IMAGE provides provides the most efficient data and I/O structure for the environments described above; VPLUS provides the most efficient method of communication with users (terminals). Both should be learned on their own, not through learning RPG interfaces.
- Knowledge of the file and operating system intrinsics is related to the success of applications.
- Efficient programs are related to efficient calculations and logic. RPG/3000 is externally driven, and there is no specific correlation between a given calculation and a specific set of machine instructions.

BRIDGING THE GAP

Understanding how to use RPG is the key to understanding how to use IBM GSD computers, because it was desgined to allow optimum use of those operating systems and instruction sets. This is clearly not the case with the HP3000. MPE was written as a language-independent (exception: SPL) operating system with independent constructs, as were IMAGE, VPLUS, KSAM and other subsystems. Obviously, our success in developing successful RPG/3000 applications lies, not as it did with IBM, in understanding MPE and its subsystems first, then learning how RPG interfaces with these things. The mistake made by the bulk of the RPG user community (at first) is to continue to rely on RPG as the window through which to peer into the computer; this is precisely what has earned us our reputation. The remainder of this paper will outline the steps involved in adopting MPE and its subsystems as language-independent constructs. Though this outline is not absolute, all of these steps must be taken in some fashion. There are no shortcuts that lead to anything other than unstable, costly to maintain systems.

1: Identify Your Resources

As early as possible, learn about any resources which are available to help you in completing the tasks outlined below. If you don't do this, being an HP RPG user will feel very lonely (let's face it, RPG is used by a minority of HP3000 customers). There are 4 sources of assistance:

HP: Read your support contract and understand what it buys you, and what is your own responsibility. If there are misunderstandings, clear them up before you proceed. Be sure your SE can help you with RPG learning and problems. He/she need not be an RPG expert to get you help. Find out who the closest RPG SE is, and arrange a path to that specialist through your SE. Learn about HP Consulting products and try to anticipate when you'll need these as you learn more about the 3000. This can be indispensable, and is also a good way to gain access to RPG specialists at HP. Finally, learn how to properly use PICS for RPG questions — an RPG specialist need not be on PICS for you to receive satisfactory response and resolution.

THIRD PARTY ASSISTANCE: When you need prolonged hand-holding and long-term help, there are sometimes third-party software suppliers who can provide help in RPG/3000 expertise. These are scarce, but nonetheless, have your sales representative check with your local third-party sales representative.

USER COMMUNITY — Locate all user groups who can provide a forum for discussing RPG-related topics, and provide a network you can call upon when necessary. HPGSUG, local RUGs, and a special interest group can all help, especially in providing you with an RPG toolbox. No special interest group currently exists. If you think it should, then help form one.

SELF TRAINING — This is probably the most important single difference between being an IBM GSD user and an HP3000 user. MPE is easily accessed by RPG users, and you can frequently solve your own problems by reading reference manuals, HPGSUG papers, etc. I have been very successful with this, and it allows you to share your solutions with others as you develop them. Again, talk to your SE to learn about all of the information that's already in your own installation.

2: Adopt MPE as a Design Determinant

Anything you do will at some point invoke MPE code. If you learn as much as possible about MPE and subsystems early, you will not be fighting with them later in debugging your RPG applications. As was stated before, this is the single most important key point in being successful with RPG/3000 — RPG calls all of the same intrinsics that COBOL, BASIC, etc. call. Here are the things you should master, along with suggested resources necessary to master them:

MPE INTRINSICS: Learn about the MPE Intrinsics — these are the basis for just about every function performed by the system. The MPE Intrinsics reference manual will provide enough information; there are sections related to Using the Intrinsics which contain good explanations of what they're useful for. Without a CALL verb, RPG can't do much with these directly, but this will still be very valuable knowledge in design and debugging. HPGSUG proceedings and HP consulting can help to solidify this knowledge.

FILE SYSTEM: Though it is actually part of the information in the Intrinsics manual, learn how the file system works. Your RPG code calls file system intrinsics for you, so you should know what you're asking MPE to do, as well as what it can do in general. Sepcifically, focus on FOPEN as it applies to RPG. This will

help you learn about the three biggest problems in RPG conversions: Buffering, Sharing, and Locking. If you understand how MPE does these things, it is much easier to ask RPG to do what you want. Again, HP consulting can be helpful here, as are issues 24 and 25 of the HP3000 Communicator.

SPECIAL CAPABILITIES: Again a subset of MPE intrinsics, two special capabilities can provide you with help in designing and converting on-line RPG systems. These are Multiple Rin (MR), which allows multiple concurrent

FLOCKS (and should be unnecessary in new systems), and Process Handling (PH), which allows your program to run another (son) program and suspend until it has completed execution.

STACK ARCHITECTURE: Learn what happens when you run a program, in terms of Code Segments, Data Stacks, Extra Data Stacks, what these terms mean in the first place (it's really very simple), and how they will affect you in the future. General reading and SE assistance will explain these things.

IMAGE: Though your converted systems will not employ IMAGE, the earlier you begin to use it, the more stable your environment will become. IMAGE is the most reliable and efficient data structure available on the 3000. Needless to say, the IMAGE course should be the first step you take, followed by RPG/IMAGE consulting, reading, and a small-scale project to let you become comfortable before you embark upon any significant new development project whcih will employ IMAGE. Converting old applications to IMAGE usually doesn't make sense, though it can be done easily and will improve your application stability.

3: Understand the Elements of Interactive Systems

Again, the choice of an HP3000 implies a change in the general approach you will be taking, and one of the most important differences is the interactive nature of the new systems you will be developing. When you offer a user an interactive system, you will need more protection against error, better recovery capability, and improved up-time. Technically, this re-alignment will require you to understand how to best use and control all peripheral equipment you will place in the hands of the user. This will involve your mastering two basic areas:

DEVICE CONTROL: Terminals and printers can be controlled directly through the use of a subset of MPE intrinsics, especially FCONTROL. Again, learn how you can control devices within the constraints of RPG. Many large systems isolate the user from MPE by using terminal monitoring and control programs which make it impossible for the user to get to a colon prompt. Though this is not possible with RPG, a terminal monitoring facility could launch RPG applications when a terminal response is requested. RPG allows you to read/write to $STDIN/$STDLIST; try all of the possible File specifications you could use to do this, and settle on one you're most comfortable with (I prefer to define an input demand and an output file). Finally, learn about escape sequences for terminal control, and all of the techniques you could use to send these to the terminal. This is easily done from RPG programs, though many RPG users are not aware of this capability.

VPLUS: Like IMAGE, a thorough understanding of VPLUS is essential to development of terminal-based RPG/3000 applications. This is probably the most controversial RPG interface, but you can be relatively successful in writing VPLUS/RPG code by following the same steps suggested earlier for IMAGE. If you try to learn RPG/VPLUS on your own and without the VPLUS class and SE consulting, chances are that you will be very frustrated, with unhappy users and unstable programs.

4: Re-Think Your RPG Design and Programming Techniques

Finally, once you've absorbed all of the material presented in the above pages, it is also beneficial to review the kind of programming guidelines you've used in the past. What you've learned about the possibilities of the HP3000 will allow you to be much more creative with your programs than you might have been in the past. Again, the following basic areas should be explored:

STRUCTURED DESIGN/PROTOTYPING: Programmers in other environments have been benefiting from two major design techniques, Structured Design and Prototyping. Do some general reading to familiarize yourself with these concepts, and determine whether either might not be of some benefit. Though interest in this technique seems to be waning, structured design does allow you to begin to start thinking in terms of small modular programs, an idea which MPE will allow you to employ easily. Modular applications allow you to develop your system as a tree of processes which you can develop, test and debug in a "top-down" fashion, which is far easier than traditional RPG development techniques. Secondly, prototyping is an idea which is becoming increasingly more prevalent because of the attendant low development costs associated with it. HP's RAPID products employ these techniques, and RPG shops who develop general programs and routines could also employ the same technique, though with not nearly the speed of development. Since RPG is a cryptic, table-driven language, it fits well with the idea of procedural brevity which is required in prototyping. Again, general reading and contact with user groups can help you learn more about these ideas.

CYCLE CONTROL: Because RPG/3000 is internally very different from IBM's RPGII, it is possible to use RPG similarly to other languages by eliminating automatic I/O (cycle driven files), and doing reading, writ-

ing, and calculations all within your calc. specifications. This is a heated argument elimination of automatic I/O does not mean you are in total control, but some users prefer this technique. Overhead in this case is not higher, as it is in IBM environments.

PROGRAM STRUCTURE: IBM indexes program efficiency to avoidance of high- overhead calculations. On the HP3000, the lowest overhead program is the program with the fewest statements and most logical calculation structure. If you use straightforward mainline code with nested subroutines, this will usually result in less object code. It will be important for you to learn about the RPG compiler internals and segmentation if this is important to you. Communicator #24 contains an interesting article related to RPG segmentation. Your SEGMENTER is the best tool you can employ to see what happens when you write a certain type of code. Be careful not to expect this to be as important as it was on IBM GSD equipment — all RPG/3000 code is not compiled, and MPE lets sloppy code execute quickly. . .

EXITING RPG: RPG doesn't take total advantage of MPE (neither does any other language); sometimes it makes sense to use the EXIT calculation to invoke a procedure written in another language. For instance, if you need to execute an MPE command from an RPG program, you could simply EXIT to a simple SPL routine which calls the COMMAND intrinsic (this could also be written in other languages), passing the command from your RPG program. This technique is almost indispensable in successful RPG/3000 systems. To learn more about this, look in the RPG reference manual, and get a copy of the REALRPG facility from the HPGSUG library, release 08. Very few HP3000 shops are monolingual.

CONCLUSION

Programming languages are simply vehicles to make computers Because of this, it is important to focus on the architecture and constructs used in the computer you're using to be successful in using a language make your computer "go fast." In the case of RPG users, we learned how to program without understanding what the programs were asking the computer to do, except in general terms. Hopefully, this paper will re-emphasize the importance of understanding the relationship of success to an understanding the HP3000 on its own terms. If the methodology outlined above is employed in an RPG/3000 installation, regardless of the age of the installation, I am quite certain that the user will be totally successful in developing high-quality interactive systems on the HP3000. As in any endeavor, attitude and organization will eventually determine how successful that endeavor is.

BIBLIOGRAPHY

1. Walmsley, David E., "RPG/3000 Programming Efficiency," HPGSUG Proceedings, September 1977, pp 42-48.
2. Todoroff, Gary, "RPG II: Report Writer of Programming Language," NOWRUG Presentation, May 1980.
3. King, David, "Current Methodologies in Structured Design," Computerworld, September, 1981, pp ID25-44.
4. Schulz, Duane, "Living in an RPG/3000 Environment," HPGSUG Proceedings, February, 1980, pp 2/75-83.
5. Schulz, Duane, Cummings, Randy, and Stevens, Brian, "RPG/3000 Application Development Course," HP SEO, Wilsonville, OR/King or Prussia, PA, December, 1981.