RTE application migration to Windows 2000

Al Morgan, Jerome Hodges, and Ellen Oliver
Strobe Data, Inc

Legacy system problem

The digital computer has safely ushered in the in the new millennium, leaving us with a stockpile of canned goods and unused generators in the basement, and a dusty 20th century computer legacy. The minicomputer revolution of the 1970s and 1980s is now dim, musty history. A new generation of technology professionals never met the computer for the "common man" -- 16 bit minicomputers which plucked the power of digital computing out of government laboratories and industrial collossi and offered it to small business, to high school labs, to small town governments. Those "affordable", "low priced" computers morphed themselves into the embedded processors we take for granted today, burrowing into weapons systems, into simulators, into automobile and aerospace test devices, into telephone switching circuits, into automated billing systems, into oil pipelines, and even into the very manufacturing processes which ran the assembly lines which produced them.

Having created the ubiquitous matrix which became the substance of the computer age, the minicomputers were unceremoniously swept ashore like so much flotsam by a wave of new generation everyman computers -- the Apple/Macintosh, the "IBM" PC, the SUN workstation, the Unix workstation. One by one, the manufacturers announced that they no longer could, or would, manufacture the old standby Data General Novas, the DEC PDP-11s, the IBM 360s, etc. And now, the venerable HP1000 will shortly add to that list.

End-of life for a computer family is not solely a determination on the part of the manufacturers that a market no longer exists for the "old stuff". It is not necessarily an admission that the "new stuff" is faster, cheaper, better", or that the paradigm shift we are experiencing today "dooms" pre-existing technology. The very components used in their manufacture are no longer made. So not only does the dreaded phrase "end-of-life" mean you can't buy the systems anymore; there will shortly come a time when they will no longer be maintainable. That presents a huge problem for the computer industry and its customers.

Billions of lines of code written for these obsolete computers represent hundreds of thousands of applications forced to find completely new computer hardware homes. This "legacy" software ranges from simple database-type applications such as bookkeeping and accounting to the brains behind complex radar systems at the heart of air traffic control systems and the multi-nation networks which predict the weather. Some of this "legacy" software runs the flight simulators that train airline pilots. "Legacy" software tests the missile engines which lift space modules into orbit. This "legacy" software still manufactures and tests modern 21st century DRAM and processor chips, not to mention the motherboards and device controllers which make eCommerce possible. Banking functions, stock trades, communications networks world-wide still depend on this legacy software. And we should not forget weapons systems developed during the Cold War, systems still maintained by every major government in the world.

Ironically, while Hewlett Packard thrusts its way into the forefront of 21st century technology, the chips fueling its new products are still being manufactured by legacy system software.

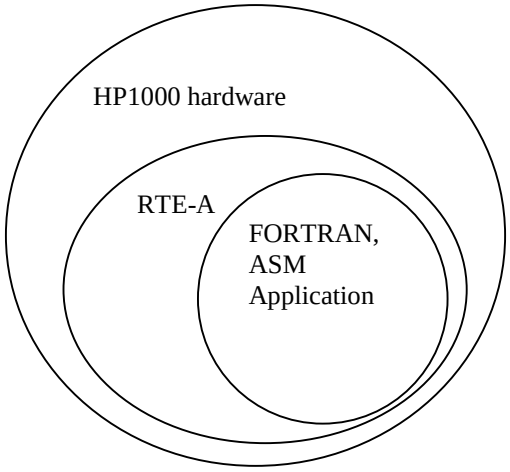
How do we transport or "migrate" this legacy software to computer hardware manufactured from modern components? The answer varies in difficulty and complexity depending on the application. We can categorize most of them as follows:

1) Custom applications which can now be replaced by off-the-shelf software
2) Simple data display systems written in compiler languages such as Fortran or COBOL which can easily be recompiled with minor changes to run on other hardware

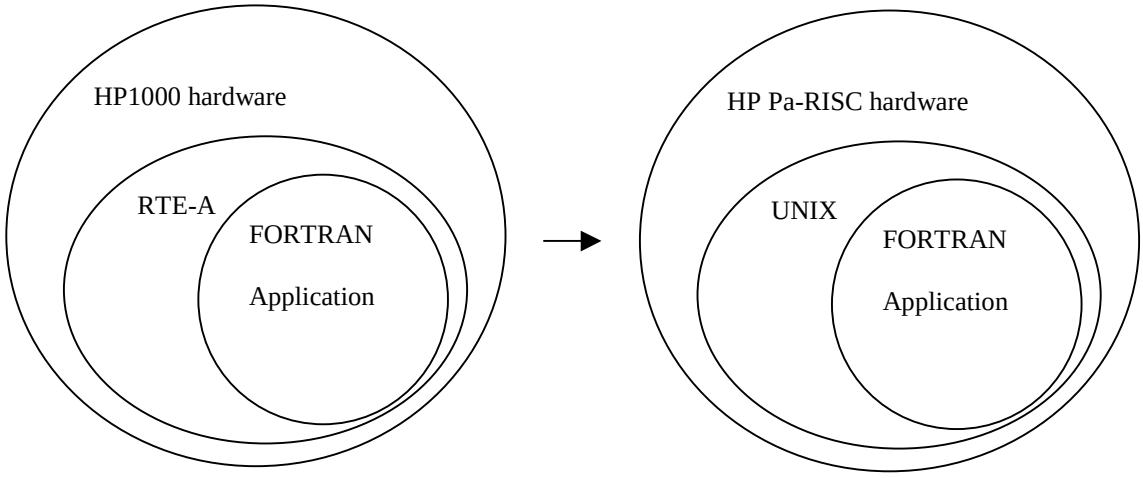
3) Database manipulation systems which can be restructured and run under database management software similar to the software used on the minicomputer
4) Scientific and statistical analysis which can be rewritten in modern higher order languages and application generators, or can be replaced by packages such as SPSS or SAS on new generation hardware,
5) Complex data analysis and display systems written around special data gathering and/or display devices not available on new generation hardware,
6) Complex networks of complex data analysis and display,
7) Large assembly language applications,
8) Time critical applications such as in process control or robotics or telemetry where the time critical problems have been solved by using special characteristics of obsolete hardware
9) True real-time applications where the computer hardware is tied very tightly to the timing of real-time responses,
10) Embedded applications dependent upon custom hardware for which no modern interfaces exist,
11) Embedded and mission critical software about which specific expertise no longer exists to assist in migration,
12) Bureaucratically certified software such as the software in commercial and military flight simulators, in-flight systems, booster engine test equipment for manned space craft, critical medical devices, etc.

What we mean by migration

Let us consider a typical custom application running on some model of an HP 1000 A-series or M/E/F series computer with a suite of peripherals. Its software "platform" is probably some version of RT6 or RTE, an operating system tailored closely and specifically to the basic hardware architecture in all its peculiar nuances. The application itself was probably written in a higher order language such as Fortran, or even ADA. Parts or even all of it might possibly have been written in Assembly Language.

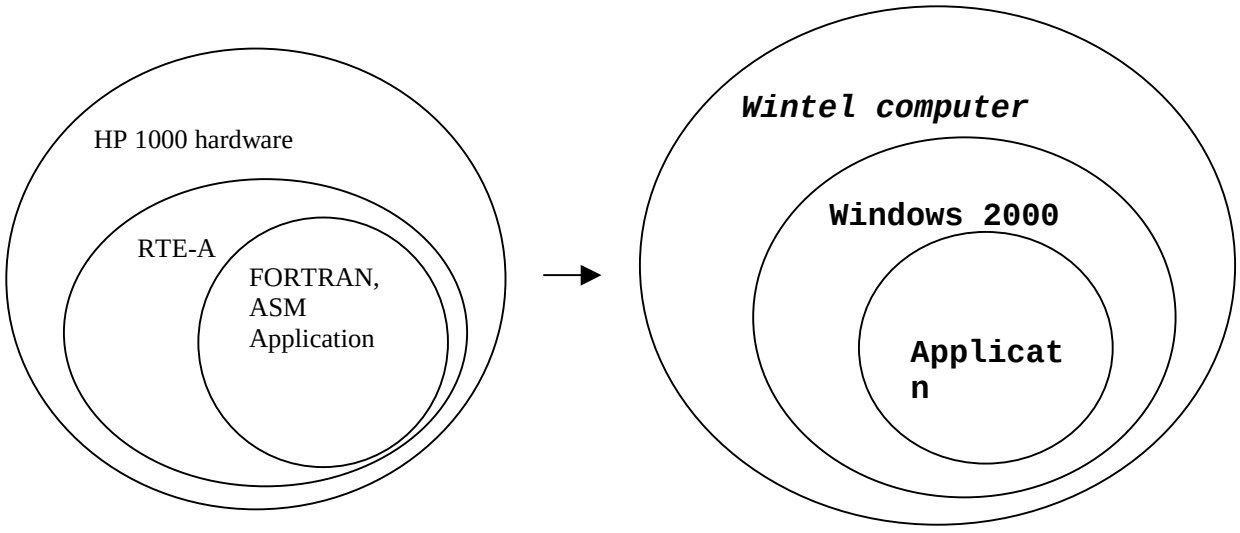


Suppose that we wish to port this application to a modern platform. Hewlett Packard offers a migration path which looks like this.



From our twelve item list above, we can find classes of applications for which this may just be the ticket. It has the advantage of completely divorcing the application from twenty-year old components; and as long as there is a reasonably long life expectancy for HP's proprietary 32-bit hardware, we are not going to have to face the migration problem all over again in the near future. HP offers assistance and suites of tools to aid in recompiling and rewriting the RTE/application interface into a UNIX/application interface.

However, for those enterprises preferring Wintel architecture and Windows, this is a difficult migration requiring more than recompiling, and if migration tools exist, they are hard to find. A certain amount of program restructuring and rewriting is doubtless in the offing just to get the legacy software communicating with Windows devices. Unless one adores writing Windows drivers and DLLs, this is not a particularly pleasing prospect, even with the help HP has recently offered with its Windows 2000 training and tools. If the application was not already in a 4th Generation package form (an Image application for example) there's going to be a lot of work to do and a lot of uncertainty about project time lines, cost, staffing, etc. As Charles Finley told us last August, migrating large legacy systems is not something for the faint hearted with shallow purses.



Migration nightmares

Not many organizations have pockets deep enough or courageous enough souls to come through unscathed. Strobe has often been the beneficiary of migration projects gone awry. We have a number of entertaining war stories to tell -- of well-intentioned people trying to prop up one failing system long enough to get some buggy package installed in its stead.

There is one classic migration example which illustrates its most dramatic pitfalls -- one that has been written about over the years, but is still not fully understood by people not close to the industry. The FAA Air Traffic Control Modernization was initiated in the late 1970s and formally got underway in the early 80s. A principal target was the radar control system based on vacuum tube generation Univacs. IBM was initially awarded the largest computer project ever funded by Congress to make the modernization happen. For decades this effort passed though contractor after contractor with incremental funding increases. The project scaled up, the project scaled down. It broke up into smaller pieces several times. But, finally, as the millenium drew to a close, the migration truly began to happen -- nearly 20 years and billions of dollars later. At one point in this 20 year period, when failing computers were making headlines at Chicago's O'Hare International, the migration engineers actually tried to put out a contract to restart the assembly line for those Univacs. Only, of course, no one could replicate them. There were no parts.

Let's look at the project from the vantage point of folk history, having had no direct first-hand experience with it, and imagine what the engineers faced. In the first place, this was a Government contract. Everything was done by the book using the most up-to-date project management tools, starting with a set of specifications.

One would think a set of specifications for an existing system would not be difficult to write. Not so when requirements creep takes over. New capabilities were added, resulting in a system more complex than the one being replaced. And that makes sense, since the new computer hardware had more capability than the old hardware, right?

Specifications agreed upon, the design phase began, including hardware interfaces. New radar equipment was contemplated. By the time the designers had a decent handle on the hardware interfaces, new hardware was proposed. That changed the specifications and changed the design and that ate up more time and resources. By the time the implementation phase came around, the computer hardware proposed was itself becoming obsolete, and that meant proposing a new platform, which meant redoing the specifications, which meant redoing the design.

It is easy to see how this project began chasing its tail in an endless round of specification, modification of specification, design, redesign, modification of redesign and then partial implementation. In the meantime, computer hardware became more powerful, capable of providing hitherto unimagined features -- features which became incorporated into the latest round of specification, redesign, re-implementation. The Mother of All Migrations had spawned the Ultimate Impossible Task bogged in tail-eating Requirements Creep.

Several lessons emerge.

- Writing new specifications for an existing system is harder than it appears. First of all, the existing system has "evolved" from its original form over a period of years and is more complex than it was initially meant to be. Secondly, it takes more discipline than most mortals possess to ignore new capability and resist the human desire to make the new version bigger and grander than the old version.
- The original systems were written in an environment were engineers had fewer resources and were forced into efficiencies which seem unnecessary today. As capability grew, features were "tacked on" becoming a part of "must have" requirements list. A system which in its day had been a large project has mushroomed into a gigantic one in danger of falling into the pit of the unmanageable.

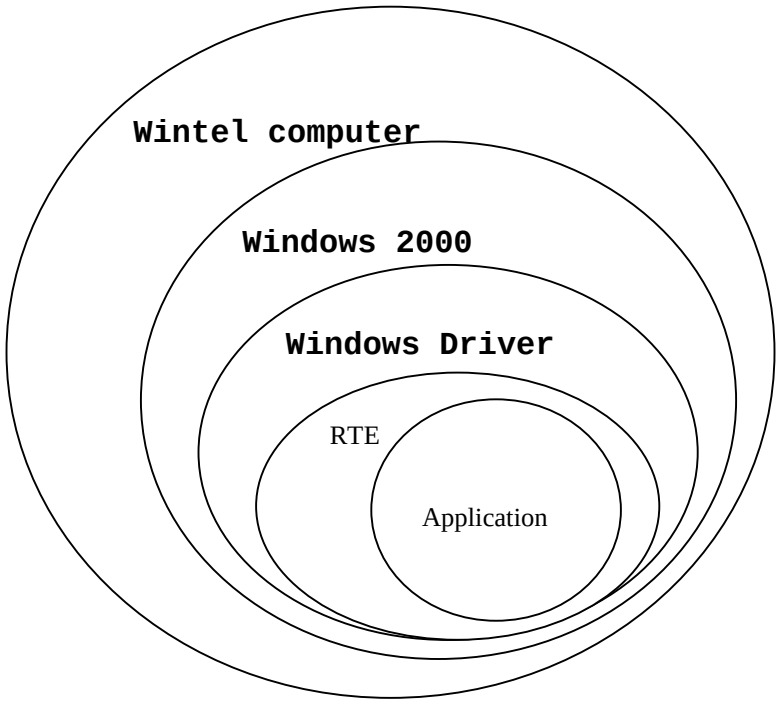
- In today's computing environment, hardware evolves so fast that it is difficult to complete the implementation of a complex system before the platform initially chosen goes obsolete. And the corollary observation is that engineers can't resist the lure of faster, cheaper, bigger, better.
- Migrating from a 1980s platform with little in common with Year 2000 architectures requires one wholesale migration step. It is difficult to accomplish piecemeal. In fact, architecting a system which integrates pieces of the old with pieces of the new can itself take so much effort that by the time you have any part of it done, you are back to the problem that the target hardware has moved on and you are still stuck with an obsolete system when you finally get done.
- Even when the migration is, for the most part, successful, retraining is expensive and emotionally stressful. When people have become accustomed to the quirks of a system that never "worked right" and are then given an entirely different user interface for another system that still doesn't "work right" (only in a different way) productivity can suffer for a long time.

We personally know MIS and project managers who ultimately lost their jobs over what seemed in the beginning to be a "simple" migration project

An "ideal" migration method

One problem overlooked when contemplating a total rewrite of the application is the psychic toll this drastic rehosting takes. It is akin to building a new Titanic in the mid-ocean while the original Titanic is approaching the iceberg. This may be possible if the Titanic is a whale boat, but can't be done for something the size and complexity of an ocean liner no matter how many people you put on the job, no matter how hard they work.

What is needed is a softer, gentler and foolproof short term migration approach. The ideal would look like the following diagram.



Suppose we pick up both the operating system, RTE, along with the application, and surround it with an interface to Windows, an interface transparent to RTE. Since the application communicates with the outside world through RTE, the application operates unchanged. If we suitably design the interface,

RTE communicates with the Windows Driver by means of RTE's existing hardware level I/O commands controlling HP1000 peripherals. If we suitably design the interface, RTE is never the wiser, as those machine level I/O commands are interpreted and passed on to Windows as PC device commands.

On the other side of the equation, Windows blindly operates normally. We have installed a complex device driver resembling the other Windows device drivers, obeying all its arcane Windows device driver rules.

Simple? Well, for the enterprise depending upon its legacy software, it's simple indeed. And in theory, at least, it allows for step-wise migration to the Windows environment by moving parts of the application piece-meal out of the RTE domain and into the Windows domain. Nothing has to be done in one grand, risky step.

Problems of the "ideal" migration method

Simple for the migrator, not so simple for the designer of this method. However, if this "ideal migration" path can be accomplished (and we at Strobe are confident it can be done since we have done it for other 16-bit minicomputers) what are the principal problems we are left with?

- **Processor Speed and throughput.** From the diagram it would appear that we have to execute two extra layers of software every time an I/O process is invoked -- RTE plus the Windows driver software. And this is undeniably true. However, even though the HP1000 A900 has a 133 nsec memory cycle time (approximately 75 MHz), the Intel architecture is a teeny-tiny bit faster -- 500+ MHz with every expectation that we have not yet seen the limits of performance increase in this architecture. Not only does this approach make use of the performance gains we read about daily, it turns out that Strobe's implementation of this "layered" approach teaches us that the Windows part of the system has so little to do that it is essentially idling. The performance issues are entirely on the RTE side where the architecture for this solution should at least match RTE performance on the A900. Our experience tells us that throughput eventually becomes dramatically greater as modern chip technology progresses.

HP 1000 A900 Specifications and Performance Figures	
Word Size	16 bits
Instruction Set	292 instructions
Time Base Generator Interrupt	10-millisecond intervals
Cache Size	4 kbytes
Cache Cycle Time	133 nanoseconds (~ 75 MHz)
Cache Fault Processing Time	532 to 931 nanoseconds
Main Memory Cycle Tim	Read: 532 nanoseconds Write: 400 nanoseconds
Average Effective Memory Access	~ 181 nanoseconds, assuming an 88% cache hit rate
Interrupt Latency (w/o DMA)	3.7 to 19 microseconds (4 microseconds typical)
Maximum Achievable DMA Rate	Input: 1.85 million words per second Output: 1.5 million words per second

- **Operator retraining.** On the surface, since the outermost layers are a Wintel computer and Windows, one would expect the operator to find himself in a modern icon-laden GUI environment much different from the old RTE/applications user interface. To be sure, the systems manager has the option of adding Windows utilities operated in the standard Windows formats to the overall application. However, the application itself runs its usual user interface through RTE, just as it always did. The PC monitor emulates the system console. PC serial ports support PCs running terminal emulations -- or retain the old terminals, if that makes sense. In short, the user interface looks the same and no retraining is needed.

- **Support for custom hardware.** This problem by itself has doomed attempts to migrate embedded systems software to new platforms and caused expensive redesign and rewriting of applications. While much of what was considered "custom" hardware in the 1980s appeared in various forms for modern platforms in the 1990s (a circumstance which allows the system manager to configure a PC with an analogous I/O properties) there still exists custom hardware too specialized to find its way into today's mass market. The ideal solution would have the PC generate the signals needed to communicate with custom hardware. In its past solutions Strobe solved this problem by including a real live controller card associated with the Windows Driver. One of the nifty characteristics of this controller card is its ability to generate the actual backplane signals of the old minicomputer. When RTE generates I/O signals for the HP1000 backplane, those signals are still sent to the custom hardware exactly as they always were.
- **I/O speed and throughput.** One lovely characteristic of the HP1000 that makes it difficult to discard is its high speed communications and I/O. Windows running on commodity systems, although gaining in this area, still cannot provide the I/O throughput of a well-stocked A-900. While the Wintel world struggles to push into the performance envelope of HP1000, the migrator can put off a commitment to Wintel compatible communications devices by making use of the backplane signal generation capability to continue to drive controllers for the HP1000 devices. In many cases, the performance disparity is an artifact of Windows itself and the demands of its re-entrant multiprocessing. By suitably configuring Windows and dedicating it to the HP1000 application, much of its I/O sluggishness disappears. An empirical note: Strobe Data has found that Windows disk caching offers by itself a significant boost in legacy system performance with some of the PDP-11 and Nova emulations. We expect a similar boost in performance here.

Interrupt Latency for Several Operating Systems			
Manufacturer	**Operating System**	**Hardware Platforms**	**Interrupt Latency**
Hewlett-Packard	HP-RT	HP 742rt, 743rt, 744rt	100 µs
Silicon Graphics	IRIX	SGI Workstations	200 µs
VenturCom	RTX for NT	x86	50 µs
Lynx	LynxOS	x86 PC, Power PC	14 µs
QNX Software	QNX 4	X86 PC	4.3 µs
Hewlett-Packard	**RTE-A**	**HP 1000 Model A900**	**~ 4 µs (3.7 - 19µs)**
Microware Systems	OS-9	x86 PC, Power PC	3 µs

(Excerpted from "An Introduction to Real-Time Computing", presented by Michael W. Thome, Digital Automation Associates, Inc., HP World 99 proceedings, San Francisco, CA and added HP 1000 A900.)

- **"Conversion".** How much work does it take to move the operating system, the applications software, and all the data from the HP1000 to the PC in a form acceptable to the Windows Driver? How can RTE boot itself, load the application, save its files without its SCSI or HPIB disk? The answer lies in the fact that RTE, through the Windows Driver, is actually reading and writing a standard PC disk. Windows disk formats are quite different from RTE formats requiring the Windows Driver to translate from one format to the other. "Empty" PC container files "emulating the HP1000 disks are created by a special utility. Then, using the HP1000 backplane signals supplied by the controller card, the user runs RTE and copies the contents of his HP1000 disks onto those container files, using RTE disk copy commands. It is a "conversion process" timed in minutes, not days, weeks or years.

For the greatest body of applications needing a migration path to and through Windows, these problems loom largest, but, as you can see, this "embedding" approach solves or mitigates them. However, a class of applications exists where the problems are not solved so readily merely by embedding the minicomputer operating system and the application in this special Windows Driver. We're talking here about real-time and/or timing critical applications.

Real-time systems present special Windows migration problems

Where data acquisition, organization and analysis is concerned, time is indeed relative. The system manager cares about the amount of time and system resources a batch process chews up. A data entry operator cares when system response seems "slow" and the computer can't keep up with her fingers. But as long as response seems humanly reasonable the data entry operator won't demand too much of a raise, and the system manager learns to become philosophic about batch execution times. No one crashes. No one burns.

Those systems we label "real time" are another story altogether. Generally, a real time system requires some kind of computer response coordinated with the real world demands of some piece of machinery. Some of that equipment is operated by humans in situations where human response time is being measured or trained. The classical example is, of course, the flight simulator. Robotics is another area requiring time-critical computer I/O. Process control systems offer another example where time (δt) is a fundamental input variable.

In a real time system, you can't mess with "time" very much and remain viable. Unfortunately, in the Windows environment, clock interrupts are subject to the same whims of Windows management as all the other interrupts. For many applications where δt is large, this is not likely to represent a crisis, particularly where one is replacing a slow CPU with a much faster one which has to sit and wait a long time for the next real time clock signal. However, there are applications where any degree of uncertainty about when the clock interrupt is likely to occur is totally unacceptable.

The real time clock

Removing this obstacle to migration requires a reliable real time clock signal and that means bypassing Windows. Strobe has solved this problem by executing time sensitive software on the controller board with a real time clock independent of time sources under the control of Windows on the motherboard.

This approach does not completely solve the problem of maintaining δt properly. One of the biggest headaches Strobe experiences with its migration solution comes about because our minicomputer replacement products end up several times faster than the old hardware. In some applications where computer programmers have inserted their own time delays the problem is not that δt gets too big for the application. The problem is that it **shrinks**. Computations which might have taken 100 milliseconds before the migration, might now be done in 10 milliseconds, and the programmer who just "knew " that there was no way the sine calculation could be finished before the expected device interrupt would occur is now in for an unpleasant surprise and THREE sine calculations are done on the same data before the next clock tick causes regurgitation of an input buffer's contents. We once saw a case where characters were fed to the screen so quickly that phosphorous didn't register them and eye never saw them. The program didn't work anymore. There was no screen display.

No matter how quickly and efficiently the embedded operating system and applications software are executed in our "ideal migration" model, it is essentially impossible to faithfully replicate the execution times of individual computer instructions. That is a given. However, the controller card gives us yet another bit of apparatus -- something to tackle the timing problem. The controller board has its own internal clock, its own crystal with a adjustable frequency. Strobe found that by allowing the end-user to change the frequency of this clock, the application and operating system can be slowed down to some point where the overall execution times of critical loops match the original well enough to allow the system to operate normally.

The case of mysteriously disappearing resources

Windows itself presents additional challenges. Vanilla Windows is flexible and general purpose. -- but this comes at a price. Windows doesn't treat the HP 1000 emulation as anything special. All bets are off: no guarantees for critical time response! If it's left up to Windows, a video game, e-mail or web surfing session can gobble up all available resources. Generous, wonderful Windows will happily host a roomful of resource stealers. So while it may be tempting to expand the operational environment of the application into the plugged in world we are accustomed to for other applications, the system configurator must provide enough discipline to preclude Internet surfing multiplexed with a solitaire session from appearing on the computer running the nuclear reactor.

Attractions of running real-time systems under Windows

The fact that the nuclear reactor operator might be tempted to use Windows' "spare" resources to play "Minesweeper", is a tribute to Windows as THE STANDARD for desktop operating systems. As such, Windows offers all of its benefits to the real-time legacy system emulation. No one argues against Windows being ubiquitous. general-purpose, and flexible.

Okay, we've said we love Windows, now let us count the ways ...

Probably the biggest attraction to legacy system owners is the price of the Windows hardware. Just consider the cost of rebuilding or even locating replacement dishwasher-sized disk drives that are pricey, slow, expensive to service and prone to failure (from head crashes, unseasoned technology, etc.). They often only have about 20 Megabytes of storage and replacement media (if available) are hard to locate. This is not to say that there are some resourceful people in business reviving units. However, the market shows that it is often more lucrative to try to adapt inexpensive, next generation hardware to work in the legacy systems.

By the way, if the hardware isn't robust enough for one's tastes, such user's cannot argue that PC pricing at it's commodity levels argues for adoption of a "disposable hardware" philosophy; replacements are so inexpensive, you can afford to throw away bad units, and quickly and easily replace the bad with new units.

Another advantage is the free speedup available as modern technology advances. Ignoring the earlier arguments about timing and the adjustable real-time clock, some RTE systems will be able to run full flat out without any slowdown. And these speedups roll up "for free" as upgraded versions appear.

Some people argue that Windows NT has always been a "half-a-step-behind" accepting the latest popular technology. But even USB is coming to Windows 2000. The fact is that the PC platform does eventually pick up the latest hardware (except for some of the gargantuan-sized technology) making those advances available to legacy software.

All the office software and suites that permeate the typical desktop PC will be seamlessly accessible to and from the legacy system, GUI, Windows and Novell networking, spreadsheet display and word processing of database software.

Summary

A Windows platform for legacy systems is a desirable way to step into the 21st century, but migration can be a nightmare.

Many organizations approach software migration as a total re-write of the application -- restructure, rearchitect, re-code. Where this can be done easily it makes a lot of sense. Where this re-write

approach cannot be done easily but where powerful migration tools are available for a new much desired host, a migration plan can be crafted with a varying degree of confidence in ultimate success at a price the organization can afford. If this migration plan takes years to accomplish, the organization would probably be much better off with something like the embedding approach described above, especially for the short term while it is finding out just how deep the doo-doo is that it is wading into.

Moving the operating system and the application, unchanged, onto a Windows platform has been accomplished for other minicomputers and it is to be expected that it can be done for the HP1000.

Real-time applications present special challenges which make them particularly difficult to migrate. The bad news is that timing requirements challenge this embedding approach as well. The good news is that even those timing problems and constraints have been overcome for other minicomputers, and, again, one can expect similar solutions will work for the HP1000.

And, finally, as Windows consolidates its position as the world's most popular operating system, the advantages of integrating legacy software into a modern Windows environment becomes more apparent to migration decision makers. Strobe hopes it will be in a position to make that decision a no-brainer.