Challenges in Application Porting and Abstraction

Table of Contents

What is Porting?

Porting is the ability to reuse software from one environment to another by making minimal changes to the existing code. Unfortunately, these minimal changes are not easy to do when it comes to porting software from one Operating System (OS) platform to another or even among versions of a single OS.

If porting requires lot of changes to the existing code, then it's called a "code re-write".

FAQs:

- Need to improve the system's functionality, usability, security, stability, and performance, which was developed sometime back.
- Need to change the system's architecture and upgrade it to newer technologies.
- Need to merge/integrate few systems with almost the same functionality that work on different platforms into one system that can run on all different platforms at the earliest.
- Need to migrate the best-selling software solution on a given system on other platforms as well.
- Need help for our best-selling software solution on an OS, as we heard that the OS vendor has stopped supporting the OS and its components.
- Need to enhance and migrate a software component to newer platforms, which was written by someone who is no longer with the company and there is absolutely no documentation or comments in the program.

The only answer to these questions is Porting and Software Reuse.

<u>Examples</u>

- Technobit saved valuable development resources and eliminated an expensive and time consuming code rewrite by using MapuSoft Technologies' OS Changer.
- QNX Software Systems used Porting to migrate their applications to QNX.

Importance of Porting

- Porting is normally quicker and cheaper to do than performing a full code rewrite.
- However, manual porting could turn into a major code-rewrite if the underlying OS platforms are very different.
- Ideally, if you could re-use your existing code without having to do any "porting" at all, then it's all the more better.
- Existing code works and has been perfected in the field. Throwing it away and starting from scratch because of a change in environments is not sensible.
- Porting is more important if there is a lot of code.

Advantages of Porting

- **Technical advantages**
  - Products can be supported for a longer life cycle.
  - It requires less engineering resources.
  - It enhances the scalability of applications on all platforms.
  - It provides versatility to the applications to run on different environments.

- **Business Advantages**
  - It saves time, money, and other valuable resources.

Porting–Technical and Business Issues

Business Issues

IT managers need to look at the infrastructure required to support the application on a new platform. This includes the human and hardware resources needed by support personnel as well as by developers. They need to hire skilled and experienced professionals in the technology to do the porting. They also require resources with a good grasp of portability issues, to deliver and handle any porting issue.

Technical Issues

It becomes important to hire or train the resources to do the porting. It is essential to make sure that the resources to be hired have the necessary skills, such as porting, testing, debugging, and performance to accomplish this task. If not, it becomes imperative to hire an efficient right third party vendor who can do the migration.

Project management Issues

It is essential to make sure that the application to be ported is properly reviewed in order to identify risks. This includes creating a schedule that allows for unforeseen events in the actual porting and testing phases**.**

Challenges in Application Porting

Technical challenges of Porting are:

- Differences in O S versions
    - Process support in VxWorks 6.x but not in 5.x
    - OS differences between 32 and 64 bit CPU
- Discontinue use of obsolete APIs
- Change code to adapt to new OS version
- Manage multiple code base across different versions
- Forced to upgrade OS due to vendor not supporting a specific OS version. As a result, also forced to upgrade hardware (from VME -> cPCI platforms).
- Application Programming Interfaces (APIs)
    - Proprietary APIs
        - Code gets locked to a specific vendor's OS platform
    - POSIX – Industry Standard API
        - POSIX Complaint versus Conformance – variations in API behavior
        - Levels of POSIX API support & what is actually supported by OS vendors
        - Vendor specific POSIX implementations (mmu/non-mmu, priority differences, hard real-time, etc.).
- POSIX is not the answer to your porting problem
- Code may compile on one compiler but not another
- Code developed in a specific language may become obsolete (Ada, Fourtran, etc.)
- Tools not available for new hardware platform
- Development on host may not be possible
- Delay in start of development until the new target hardware/tools become available
- Need to deal with target issues while porting

- Longer-term porting solution:
  - Develop common OS interface APIs that your code can reuse across multiple OS & their versions, otherwise called an in-house OS abstraction
  - Check out the commercial OS abstraction solutions

Porting Tools

Porting tools are used to port a program/application to another OS or another platform.

The best way to start your porting effort is to use the best available tools that can reduce your efforts and maximize your time to market. These tools have an option to begin the development on host so that the developers need not worry about the underlying OS. They support a wide variety of OS's that any company is looking for porting the applications.

The tools run in simulation and debugging modes. They have a proven architecture. Postmortem analysis of your code is possible with tools such as Profiling and Performance monitoring.

You may have to provide the source code to a third-party if you utilize them to do the port. Do it yourself by using a tool.

Hosts provide development tools (IDE's) which are more friendly tools such as compilers, project management, C/C++ syntax editors, simulators, and debuggers. Some tools have different compilers for the same architecture or language. The customer can customize the features and functions according to their needs.

> **Example:** MicroC has different compilers TINY, MEDIUM, HIGH which give different foot prints based on the features the customer wants.

Some host development tools provide state-of-art graphic development tools. Host development can have high quality development environments as there are only few platforms the tools have to address.

Target boards do not have standard output devices to display printf. Other output devices exist but not at the project startup. This is the same with input devices also. Some times development starts with a processor and a clock with few peripherals. Most of the peripherals and I/O devices such as Ethernet, JTAG, serial/parallel/USB ports are added later. They are added after checking that the processor is working fine and with all its core functionality.

Target boards have less memory. This makes it difficult to look for the code foot print. This is not needed if you are developing in the host. If the target board is completely new, it takes time to port the OS. This gives you ample to time to develop the applications in the host.

These tools make development possible even if there is no hardware available. Debugging the code is very difficult if the code hangs in a loop and so on. Tools are more robust on host systems.

There is no need to worry about memory or performance issues in host development as there are abundant resources available in the host.

Using a Porting Solution provider

They provide end-to-end migration of the application software.

The main advantage of end-to-end migration providers is that the internal resources are not diverted from the primary responsibilities of software development.

The disadvantages are:

- There is sharing of source codes and risk on the IP generated for long.
- This requires training the customer on the functionality and product, which involves time and money.
- This calls for additional project management and there is no control over project schedules as there are external dependencies.

What is Abstraction?

Abstraction provides the ability for software to be re-used from one environment to another with no changes to the existing code

Without an OS abstraction, it would be difficult to move software across OS platforms and their versions.

Examples

- The Boeing used OS Abstractor solution System of Systems Common Operating Environment (SOSCOE) architecture
- Integrated Defense Systems(IDS)
- IBM
- STMicroelectronics used VxWorks I/O interfaces instead of native Nucleus interfaces

Importance of Abstraction

Abstraction is essential to:

- Avoid porting issues and protect the code against future platform changes.
- Ensure that the fundamental OS resources behave the same across all platforms.
- Ensure that there will be no impact on system performance.
- Make it cost-effective in terms of time, money, and resources to build and maintain an in-house abstraction for multiple operating systems.
- Avoid shifting focus from the organization's engineering and core competencies.
- Reduce potential learning curve while working with the new Operating Systems and make the code reuse easier to adopt.

Advantages of Abstraction

Abstraction ensures all applications run on different OSs and abstracts all the APIS's in the OS.

Abstraction can be done on data types, header files, and symbolic defines. It allows new OS support quickly and seamlessly.

Abstraction makes easy transition from/to any OS, leveraging on the existing software and knowledge base.

It develops proof-of-concept demo software quickly. This also helps customers change from proprietary to open source OS.

Abstraction makes porting easy, simple, and less time-consuming task.

Challenges in Abstraction

It is difficult to keep track on the OS versions that the system may use, as that would require predicting the future in order to plan for all changes to the application's requirements.

It is difficult to predict if the operating systems planned for future will always be available.

It is difficult to plan for the new and future OS versions.

In order to accommodate future requirements and avoid any such issues, developers need a COTS OS abstraction to make developing portable software effortless.

OS Abstractions should not impact the application's performance. Abstraction should not waste valuable CPU resources. It should use compile time translations. It also should utilize low level high performance APIs.

Abstraction needs to eliminate dynamic allocation of resources (pre-allocate) and should be fully scaleable. It should have a small footprint.

OS Abstractions should not fully rely on the underlying OS. OS Abstractions should add missing API features.

The abstraction needs to be flexible for design alterations without requiring a software rewrite, thus protecting the software investment.

The abstraction tool should be engineered with safety critical features vital to defense and mission systems, without sacrificing real-time performance.

Typical Abstraction Layer

Figure 1 represents a typical Abstraction Layer.

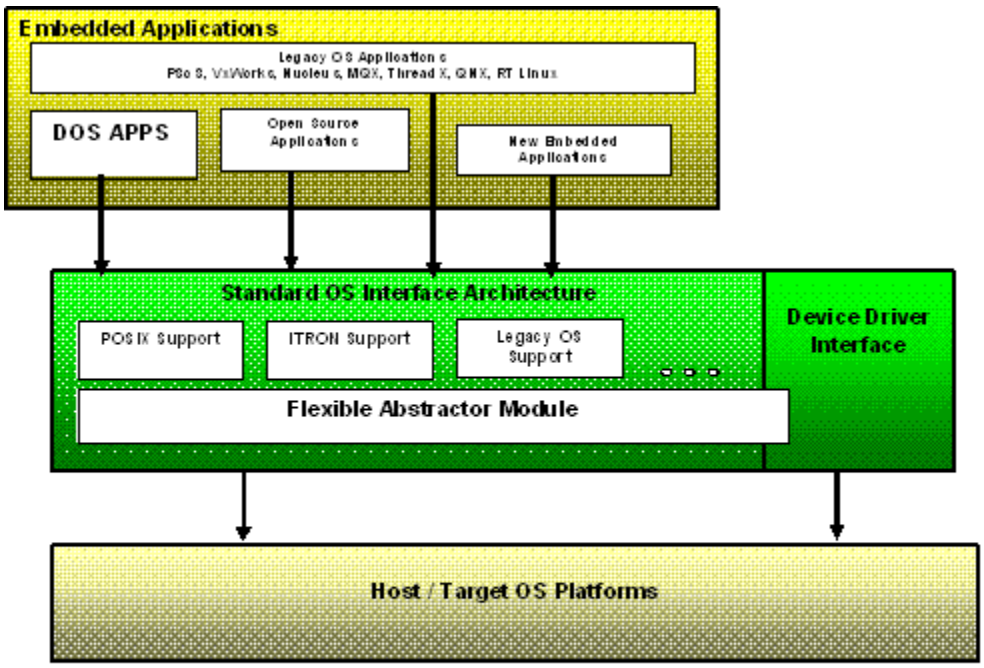

Figure 1: Abstraction layer

Introduction to Software Reuse

1. Software reuse is a critical strategy for all software development groups.
2. It is vital to systematically reuse code rather than throw away the investment and start from the scratch.
3. It is important to fully implement code reuse throughout the organization.
4. Reuse code to leverage on the existing software while moving to the next generation platforms.
5. Code reuse provides maximum benefits to an organization if it is done systematically, rather than sporadically and opportunistically.

According to Douglas C. Schmidt, Associate Chair of Computer Science and Engineering and Professor of Computer Science at Vanderbilt University,

"Systematic software reuse is a promising means to reduce development cycle time and cost, improve software quality, and leverage existing effort by constructing and applying multi-use assets like architectures, patterns, components, and frameworks"

Resources: For more information, go to http://www.cse.wustl.edu/~schmidt/reuse-lessons.html.

Importance of Software Reuse

- Software reuse changes the system's architecture and upgrades it to newer technologies.
- This helps to integrate multiple systems into one system that can run on different platforms.
- Software reuse helps to migrate best selling software solutions to other platforms.
- This supports the OS when OS vendors fail to support the older versions of the OS and their components.
- This enhances the process of migration of software components to newer platforms.

Advantages of Software Reuse

- Software reuse increases software productivity and interoperability.
- It develops software with fewer resources.
- It reduces software development time and maintenance costs.
- It produces more standardized and better quality software with a powerful competitive advantage.

Challenges for Software Reuse

The most critical points of concerns of software reuse are:

- The third-party dependencies of the application. In some rare cases, the third-party products that the application uses may not be available on the target platform, or they are only supported at newer versions.
- Finding this critical point during the assessment and development phases makes this as a risk to the porting effort.
- Just as reusing code on different operating systems, reusing code when moving to a different language presents difficulties as well.

    For example, many companies are now moving away from the Ada language to the more modern C language, due to a lack of programmers and support for Ada. Here organizations are utilizing COTS language conversion tools for automatic conversion to avoid a rewrite.

- On the non-technical side, while top level executives and government agencies only see the benefits of code reuse, there is a lack of goal congruence with engineering groups and subcontractors. Many times theses groups have psychological barriers to reusing code.
- However, by reusing the legacy code quickly and efficiently with COTS code reuse solutions; they can contribute to new projects and product development, rather than being bogged down by wearisome porting work.
- Organizations may also need to change productivity policies and benchmarks to effectively integrate code reuse into their culture. Rather than focus on how many new lines of code the developer contributed.
- This reduces the time for project completion which motivates engineers to use COTS porting tools to meet an earlier deadline. This leads to more projects, more new products, and ultimately more opportunities to get a larger market share in the organization’s industry.

Other software Reuse Tools

- Host Development
  - Testing and simulation
  - Begin developing applications before target hardware is available
- Cross-OS Driver development
- Profilers
  - Test and debug the code
  - Run code performance analyses
- Code conversion tools
  - Ada to C/C++
    - This can be especially helpful in keeping older military applications up-to-date.