8/24/17

True Meaning of the Platform Independence

Most famous language of all that claims platform independence is Java. We don't have anything truly platform independence in currently technology. This may raise a lot of Java geeks to jump in and say, "You don't know what you are talking about."

My definition for true platform independence is a language that all machines really understand WITHOUT any trick. Trick is the key word here. The trick of the Java is their Run Time Environment. It's just a program specifically precompiled for various machines (in different particular versions) so that they can understand Java Codes. Without it, no machine in current technology will understand any language but their manufacturer's assembly language. We don't even have universal assembly language at our present time that can program any unknown CPU. Our current popular CPUs made by Intel Corp, AMD Corp, and others are preprogrammed to understand their very particular unique low-level assembly language. Someone may be able to play around to figure out which of the commands of the unknown assembly language a particular language does understand and not, but there is NO assembly language that can talk to ANY CPU and CPU will understand all the "words" it is saying to it...

Many people do not know that CPU manufacturers designed their CPUs and had to create a picky particular language for each version or type of CPU they manufacture. After this, they have to work with popular OS companies like Apple, Microsoft, and IBM in order for their CPUs to be able to understand a particular Operating System. Note, IBM manufactured weird CPUs that only their Assembly Language and their OS can understand; they have their proprietary or secretive unique designed CPUs as well as publicly transparent CPUs… BTW, I think manufacturing these secretive systems from scratch CPU architecture, assembly language tasks, giving unique OS, and control of the usage of the machine to the end make IBM a struggling IT company.

To continue with Java, what they claim "platform independence" is that once the Java program is "compiled" you can run that complied program from any machine that has the Java Run Time Environment installed. No one in the world seems to argue, however, that Java Run Time Environment is a particular compiled program itself that has to be made particularly for various machines. If Java language is truly platform independence, there should be only one Java Run Time Environment that could be installed the same on all versions or Windows machines, Linux machines, Apple machines, IBM machines, and everything else. They shouldn't have to make (or compile) those Run Time Programs for different machines discriminately. To take the argument further, the language shouldn't even have that cheesy trick (a lie) called Run Time Environment, if there is a software language exists that claims it is smart enough to figure out how to talk with CPUs.

Questions remained for me, however, that there exist unique “in-house” made CPUs from big scientific and government institutions like CPUs that control radar systems, satellite systems, nuclear power plants, rockets, and etc that use Java language. Many of them used Java, but Java Run Time Environments aren’t commonly available for those uncommon systems. First option, I think geeks in those institutions designed their systems from common CPUs, programmed their CPUs with known or close-compatible assembly language, modified the common operating systems to their desired state, and then installed the common Java Run Time Env so that their systems can now be programmed using high-level Java language. I wonder if those big institutions actually paid Oracle or old Sun Micro Systems specially to write special Run Time Env for their uncommon systems… I know from my personal experience that many geeks working at Rome, NY Air Force Research Lab do not talk to or do any special business with the owner of Java language. In fact, they violate Java’s term of use, explicitly forbidding anyone to use Java for weapon systems, and nuclear systems...

Another possible option for them is to crack the Java Run Time Environment technology and compile their own unique Java Run Time Env that will interpret the real Java language. It is very possible and probably easier than the first option. Many people said Microsoft did that and thus C# was born. There were historic legal dispute cases about Microsoft stealing Java secret tricks and inventing C# out of it...

Yet another (possibly the last) option would be to manufacture the CPU that can understand the direct high-level Java language without the non-sensible so-called JVM machine and that Run Time Env. This would be the most difficult and challenging cool task but I imagine it’s quite possible. Microsoft and several other known establishments have cracked the Java compiler’s tricks and turned what they learned into a “different” languages like C#, JRuby, JPython, Junit, Android Development Env, and etc. Why not implement all those compiler’s rules into a hardware CPU’s rules or grammar environment instead of a software? This could be done directly at CPU hardware architecture level, or just an assembly language programming it to facilitate some parts of hardware’s difficulties to compliment with software to talk with Java directly. This is different from option 1 because it will bypass the operating system entirely and this system doesn’t have an operating system. Rather Java language itself would be its operating system…

Still would those three options above be considered “Platform Independence” because Java could now talk to the CPUs directly? Nope! Those are just special made CPUs so that they can understand Java talking to them directly. Java won’t be smart enough to go and talk with “ordinary” CPUs. As far as I understand there aren’t CPUs made these days that can understand Java directly without someone tweaking them. Those options above would be costly IT engineering tasks! I heard however that oracle is persuading Intel and AMD to preprogram their CPUs arithmetic bite-code architectures to resemble those of Java. The CPUs may not have the whole capability of a true compiler, but having some memory allocation the same in the arithmetic systems as Java would speed up the Java run times extremely fast—I imagine. I will leave this area loose and I don't want to verify and clarify on this topic. I read about this somewhere sometime ago, but I will end this article here now. Please feel free to discuss further. Especially, enlighten me and fill me in with some missing or inaccurate info or idea.