

**PRE-BOOT INTERPRETED NAMESPACE
PARSING FOR FLEXIBLE
HETEROGENEOUS CONFIGURATION AND
CODE CONSOLIDATION**

TECHNICAL FIELD

This disclosure relates generally to using an interpreted language code to interact with hardware devices of a processing system during a pre-boot runtime, and in particular but not exclusively, relates to sharing advance configuration and power interface machine language control methods across a pre-boot runtime and an operating system runtime of a processing system.

BACKGROUND INFORMATION

Modern computers are complex computing systems, evolving at an ever-increasing rate. With rapid evolution of technologies, original equipment manufacturer (“OEM”) system builders are presented with the difficult task of providing seamless integration between cutting edge technologies and legacy technologies. As a result, these OEM system builders often resort to ad hoc methods to integrate the new with the old. These ad hoc methods, while often providing a sufficient solution, often fail to fully leverage the advantages of these new technologies.

One such new technology is the Advance Configuration and Power Interface (“ACPI”), defined in the ACPI Specification, Revision 2.0a, Mar. 31, 2002 developed in cooperation by Compaq Computer Corp., Intel Corp., Microsoft Corp., Phoenix Technologies Ltd., and Toshiba Corp. The ACPI Specification was developed to establish industry common interfaces enabling robust operating system (“OS”) directed motherboard device configuration and power management of both devices and entire systems. ACPI evolves an existing collection of power management BIOS code, Advance Power Management (“APM”) application programming interfaces (“APIs”), and the like into a well-defined power management and configuration interface specification. ACPI provides a way for an orderly transition from existing legacy hardware to ACPI hardware, and allows for both ACPI and legacy mechanisms to simultaneously exist within a single processing system.

The ACPI specification further describes a programming language, called ACPI Source Language (“ASL”), in which hardware designers can write device interfaces, called control methods. ASL is compiled into ACPI machine language (“AML”) and the AML control methods placed in ACPI tables within system memory for use by the OS to interact with hardware devices.

The basic input output system (“BIOS”) sets up the ACPI tables during the boot process (i.e., pre-boot runtime); however, the BIOS itself does not use the AML control methods to interact with the hardware devices of the processing system. Instead, the BIOS relies on BIOS APIs, generally stored in nonvolatile flash memory, to perform the very same interactions with hardware devices as are described by the AML control methods. These BIOS APIs are usually coded in C and compiled into machine language binaries for use by the BIOS.

Thus, OEM system builders must include two independent sets of coded device interfaces—APIs for use by the BIOS and AML control methods for use by the OS—to perform the same tasks. This ad hoc integration of the new ACPI technology with the old BIOS API legacy is wasteful both in terms of limited nonvolatile flash memory and OEM

system builder time. Furthermore, this ad hoc integration fails to fully leverage the advantages of ACPI.

For example, AML is a declarative language which describes how a particular interaction with a hardware device may be accomplished and allows the entity calling the AML control method to decide whether or not it wishes to execute the particular tasks described. AML increases system reliability by bounding and guarding the operation of low-level management code. In other words, AML is transparent as to its internal or physical level operations. In contrast, BIOS APIs are defined by an imperative machine language called binaries. An entity calling a binary has no idea how or what the binary executes to accomplish the requested hardware task. Because BIOS APIs are manipulating hardware registers to control hardware devices, the computing system is particularly vulnerable to errant writes and other failures. Prior experience with BIOS APIs shows that they are a rich source of problems. Thus, API binaries do not provide the level of supervision and transparency of operation, as provided by AML control methods.

Another deficiency with API binaries is their lack of portability between software platforms. API binaries are compiled to execute within a particular platform environment. Where as AML control methods abstract the physical implementation through use of an OS interpreter. The OS interpreter interprets the AML control methods on the fly thereby accommodating various software platforms. The OS interpreter (a single entity) may need to be platform specific, but the multitudes of AML control methods are platform independent.

BRIEF DESCRIPTION OF THE DRAWINGS

Non-limiting and non-exhaustive embodiments of the present invention are described with reference to the following figures, wherein like reference numerals refer to like parts throughout the various views unless otherwise specified.

FIG. 1 is a block diagram illustrating a processing system to execute interpreted language code to interact with hardware devices of the processing system during a pre-boot runtime, in accordance with an embodiment of the present invention.

FIG. 2 is a flow diagram illustrating a method to execute interpreted language code to interact with hardware devices of a processing system during pre-boot runtime, in accordance with an embodiment of the present invention.

FIG. 3 is a block diagram illustrating an ACPI namespace for sharing AML control methods across the pre-boot runtime and the OS runtime, in accordance with an embodiment of the present invention.

FIG. 4 is a flow diagram illustrating a method to execute interpreted language code for configuring hardware devices during a pre-boot runtime, in accordance with an embodiment of the present invention.

FIG. 5 is an exemplary setup display for configuring hardware devices using an interpreted language code during the pre-boot runtime, in accordance with an embodiment of the present invention.

FIG. 6 illustrates an exemplary computer system to execute interpreted language code to interact with hardware devices of the computer system during a pre-boot runtime, in accordance with an embodiment of the present invention.