

SELF-DESCRIBING ARTIFACTS AND APPLICATION ABSTRACTIONS

TECHNICAL FIELD

This invention generally relates to a technology for managing the software components of one or more computer systems.

BACKGROUND

The reliable installation, configuration, maintenance, and removal of components of a conventional software-based computer have long been a difficult problem. Examples of such components include the following: operating systems (OSs), application programs, device drivers, application programming interfaces, and other systems programs. Examples of conventional software-based computers include the typical general-purpose personal computers (PC) running one of the many existing OSs.

A software-based computer is typically embodied by the persistent contents and configuration of its secondary computer-storage system (e.g., a "hard disk"). The conventional computer-embodiment content and configuration is merely a collection of bits accumulated over time and without centralized oversight and coordination. Typically, these accumulated bits are the result of a series of individual ad hoc events throughout the lifetime of the computer. Examples of changes include, for example, installation of a program, change of a configuration setting in a registry key, deletion of a file, or installation of a software patch.

When a software-based computer boots, it merely executes whatever the computer finds on hard disk. Since the correctness of the contents on the computer's disk ultimately depends on the correctness of each of these ad hoc events over the lifetime of the computer, the contents and configuration of the computer may readily become corrupted, damaged, skewed, obsolete, or simply incorrect.

The correctness of the computer's contents and configuration is further threatened by other externally initiated ad hoc event involving a malicious attack by a virus, a worm, spyware, and the like. Unbeknownst to the user of the software-based computer, these malicious attacks alter the computer's contents and configuration, most likely in a manner that is inconsistent with the user's desires.

Various products and services (e.g., so-called "anti-virus" and "disk cleanup" utilities) are available for detecting and correcting a computer's contents and configuration that have become corrupted, damaged, skewed, and/or attacked. While clearly well intentioned, these products and services may just compound the problem by introducing yet another ad hoc event to the resulting accumulation of bits on the computer's disk.

Conventional software-based computers are inherently brittle. One reason is because the computer's collection of accumulated bits has incomplete descriptions that are, at best, anecdotal. These incomplete descriptions are merely the results of the same series of ad hoc events and do not systematically describe the bits on the disk or the series of events that produced them. They are also unmatched with any specification, total or partial, of what the system configuration should be, or of any way of checking the state against the specification.

The following fact illustrates the inadequacies of conventional software-based computers: Given an arbitrary offline "system image," one cannot in general determine conclusively that the system image contains a functional OS or a

specific functional application. A system image is a bit-for-bit copy of the contents and configuration information typically persisted on a hard disk of a conventional software-based computer. Those contents and configuration, as discussed above, embody the computer.

Given a system image, one may check if specific files exist on the image. This check may be done with empirical knowledge of which specific files are installed with a particular OS or particular application. However, such empirical evidence does not tell one whether all of the necessary components (of the particular OS or particular application) are installed. Such empirical evidence does not tell one whether there are any conflicting components are installed. Further, such empirical evidence does not tell one whether all of the components (of the particular OS or particular application) are configured correctly to produce a functional computer. Such checks are necessary but not sufficient.

Even if one empirically determines the existence of all of the specific files necessary for a particular OS or application to function, that fact is not sufficient for one to know that particular OS or application on the image will function correctly. Again, these checks are necessary but not sufficient.

Indeed, the only effective conventional recourse is to abandon the offline examination of the image and implement an online examination. One may boot a computer using the system image and observe the results. This conventional approach is often impractical, unsatisfactory, and unsafe. Clearly, this approach is not scalable.

Even using the conventional approach of an online examination, it is often difficult to identify, without doubt, which particular applications and/or OS components are installed or even currently running. Often all that one has determined is that an application or component having a specific name exists on the computer. This determination relies on the software developers avoiding the use of misleading names. Such misleading names may occur inadvertently or purposefully.

For example, while existing OSs available in the marketplace (like Windows® XP or Linux) might show a list of running processes, by name of the file used to start the process, the name of each running process is only a "hint" as to true identity of the process. For example, an innocuously named process might have been hijacked by a virus and subverted to another potentially malevolent task. Alternatively, a properly named process may have been corrupted by, for example, an administrator installing a seemingly unrelated application.

With conventional software-based computers, there is no descriptive structural link between low-level software abstractions and high-level software abstractions. This means that there is nothing in the structure or architecture of the conventional software-based computers that descriptively and necessarily links low-level and high-level software abstractions.

Low-level software abstractions include, for example, particular files (e.g., load modules) on a disk and particular processes executing on the computer. High-level software abstractions include, for example, applications programs (e.g., Microsoft® Publisher® desktop publishing product) and families of applications (e.g., Microsoft® Office® suite of office productivity products).

For example, the concept of an application program is part of a user-centric model. Where a user sees an application program (or group of programs) that helps the user accomplish a specific task (e.g., word processing, spreadsheet analysis, and database management), a conventional software-based computer merely sees one or more active processes. There is nothing inherent in the architecture of the