

manifest to determine which applications will be invoked soon, the self-describing artifact manager can encourage the start of some applications before they are actually invoked.

[0099] At 216, the execution gatekeeper 162 examines the associated self-describing artifacts to determine whether to allow execution of the associated application (or other program) based upon the current conditions and the declarative descriptions of the associated manifests. When such determination is made, the gatekeeper may limit or prevent execution of the associated application (or other program).

[0100] For example, local policy of a computer may precisely describe which applications can and cannot be invoked, as well as the manner in which they may be invoked. If so, the gatekeeper will only allow invocation in the specified manner.

[0101] At 218, the system verifier 164 audits the integrity of the system against external modification. The audit is based upon the manifests of the self-describing artifacts of the system. For example, a load module's manifest may contain a signed digest of the contents of the one or more associated load modules. The gatekeeper can periodically check the contents of all load modules to see if they still match their specified digests.

Methodological Implementation of Exemplary System Verification

[0102] FIG. 3 shows a method 300 performed by the systems verifier 164. This methodological implementation may be performed in software, hardware, or a combination thereof. For ease of understanding, the method is delineated as separate steps represented as independent blocks in FIG. 3; however, these separately delineated steps should not be construed as necessarily order dependent in their performance. Additionally, for discussion purposes, the method 300 is described with reference to FIG. 1.

[0103] At 310 of FIG. 3, the systems verifier 164 responds to a triggering event. Examples of a triggering event includes (by way of example and not limitation) receiving a manual verification request, performance of action by another program (e.g., installing software), and a schedule time event. This triggering event may be identifiable and associated with a particular type of desired verification.

[0104] At 312, the system verifier 164 examines the manifests of the self-describing artifacts to gather information from those manifests.

[0105] At 314, the verifier performs a verification of the online and active system of the computer 102. More particularly, it is a verification of the self-describing artifacts. As it is possible, these verifications may be performed on an offline "system image" as well.

[0106] The verifications performed by the verifier are designed to promote the stability, integrity, robustness of the system in its fully functioning condition. The following are a list of example verifications that may be performed by the system verifier 164 (list provided by way of example not limitation):

[0107] verifying that that all dependencies of installed software in the computer 102 are met;

[0108] verifying that an operating system of the computer 102 includes all device drivers necessary to run on the hardware configuration of the computer 102;

[0109] verifying that code in the computer 102 has not been altered either accidentally or maliciously;

[0110] verifying that an application is correctly installed in the computer 102;

[0111] verifying that a known faulty or malicious program is not installed on the computer 102;

[0112] verifying that an application and all of its constituent components and dependencies exist on the computer 102 before installation;

[0113] verifying that an application is installable on the computer 102 before loading its components onto a system;

[0114] verifying that installation of a new application or system component will not conflict with existing applications or components;

[0115] verifying that an application or operating system component can be removed without breaking dependencies from other applications or components;

[0116] verifying that an application or operating system conforms to a predefined local policy.

[0117] At 316, the verifier reports the results of the verification to whatever called it (e.g., the OS 112 and/or the user).

System Inspection

[0118] This new architecture overcomes many of the inadequacies of conventional software-based computers. For example, given an arbitrary offline "system image" of a software-based computer using the new architecture, one can, indeed, determine conclusively that the image contains a functional OS or a specific functional application. This cannot be done with a software-based computer using a conventional architecture.

[0119] The manifest of each artifact is stored within (or can be retrieved along with) a persistent "system image" of a software-based computer's content and configurations. The artifacts are stored in such a manner that their associated metadata (of their manifests) can be inspected (by, for example, the system inspector 180) when the image is offline. In addition, other contents of the image may be inspected as well.

[0120] In doing so, the system inspector 180 may make strong statements about the contents and the future behavior of the system. This is possible even if the metadata and other parts of the system image are scattered across a distributed store, such that they come together only when the system boots and runs. By examining the static manifests of the self-describing artifacts and proposed or anticipated dynamic manifests of associated prototypes and abstractions, the system inspector 180 can verify a number of properties of the software-based computer 102.

[0121] For example, the inspector can verify the classes of properties supporting compositional verification. In other words, the inspector can determine whether all of the necessary elements of a component of the software-based computer 102 exist on its persisted system image and that the elements compose correctly.