

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).

[0016] 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 conventional software-based computers that descriptively and necessarily links the active processes (and their load model sources) with the representation of the application program that the user sees (typically via a graphic user-interface (GUI) process).

#### SUMMARY

[0017] Described herein is at least one implementation employing multiple self-describing software artifacts persisted on one or more computer-storage media of a software-based computer. In this implementation, each artifact is representative of at least part of the software components (e.g., load modules, processes, applications, and operating system components) of the computing system and each artifact is described by at least one associated “manifest,” which include metadata declarative descriptions of the associated artifact.

#### BRIEF DESCRIPTION OF THE DRAWINGS

[0018] The same numbers are used throughout the drawings to reference like elements and features.

[0019] **FIG. 1** shows an example operating scenario for an implementation described herein.

[0020] **FIG. 2** shows a flow diagram showing one or more methodological implementations, described herein, for management of persisted self-describing artifacts and performing gatekeeping on execution of software components composed of, at least in part, of the self-describing artifacts.

[0021] **FIG. 3** shows a flow diagram showing a methodological implementation described herein to verify the persisted self-describing artifacts.

[0022] **FIG. 4** shows a flow diagram showing a methodological implementation described herein to inspect an offline “system image” composed of, at least in part, of the persisted self-describing artifacts.

[0023] **FIG. 5** is a diagram showing an example interrelationship structure amongst software components (e.g., load modules, processes, applications, and operating system components), the example structure being in accordance with an implementation described herein.

[0024] **FIG. 6** shows a flow diagram showing a methodological implementation described herein to create and manage application abstractions.

[0025] **FIG. 7** is an example of a computing operating environment capable

#### DETAILED DESCRIPTION

[0026] The following description sets forth techniques implementing a computing technology for a software-based

computer employing self-describing software artifacts. An exemplary implementation of these techniques may be referred to as an “exemplary self-describing artifact architecture.”

[0027] The exemplary self-describing artifact architecture provides a refreshing and invigorating approach to the realm of computer science. Rather than being no more than an accumulation of bits resulting from series of ad hoc events during the lifetime of a software-based computer, the contents and configuration of the computer utilizing this new architecture is an organized, stable, reliable, robust, and deterministically constructible collection of self-defining software artifacts.

[0028] Before describing the new architecture, a brief introductions of terminology is appropriate. The following terms, as used herein, are briefly defined here. However, the reader is encourage the read the full text to understand and appreciate the full meaning of each term in the context of the full description.

[0029] Software Artifact (or simply “artifact”) is an offline manifestation of an executable entity (e.g., a process, an application, a component of the operating system); it includes, for example, load modules and configuration files.

[0030] Manifest is metadata declarative description of an executable entity. A manifest may be associated with each manifestation of an executable entity. Manifest may be static or dynamic.

[0031] Prototype is an executable (or “runable”) manifestation of an executable entity, but a prototype of an entity is not in an executing state.

[0032] Abstraction is a manifestation of an executable entity when it is in an executing state (“it is running”).

[0033] Component is a part, portion, or constituent element of a manifestation of an executable entity; For example, an application includes process components and a process includes executable instructions as components.

#### Exemplary Self-Describing Artifact Architecture

[0034] **FIG. 1** illustrates one view of an exemplary self-describing artifact architecture **100**. In this view, the architecture **100** is implemented on a software-based computer **102**, which is configured with a memory **110** (e.g., volatile, non-volatile, removable, non-removable, etc.). The computer **102** has an operating system (OS) **112**, which is active in the memory **110**.

[0035] The computer **102** has access to at least one computer-storage device **120** (e.g., a “hard disk”). The computer-storage device **120** contains the contents and configuration that embody the computer **102**. The contents include various software components, which include (by way of example and not limitation) an operating system (OS), the OS elements, all installed applications, and all other associated components (e.g., device drivers, installation files, data, load modules, etc.). The configuration includes the specified properties of each software component and the defined interrelationship amongst the components.

[0036] For the purposes of this discussion, references to the “system” represents the software-based computer **102** as