(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
18 September 2014 (18.09.2014)

WIPO | PCT

(10) International Publication Number
WO 2014/145132 A1

(51) International Patent Classification:
G06F 9/30 (2006.01) *G06F 9/06* (2006.01)

(21) International Application Number:
PCT/US2014/029839

(22) International Filing Date:
14 March 2014 (14.03.2014)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/798,362 15 March 2013 (15.03.2013) US
13/925,697 24 June 2013 (24.06.2013) US

(71) Applicant: **WESTERN DIGITAL TECHNOLOGIES, INC.** [US/US]; 3355 Michelson Drive, Suite 100, Irvine, California 92612 (US).

(72) Inventor: **BRIDGES, Dick**; 3355 Michelson Drive, Suite 100, Irvine, California 92612 (US).

(74) Agents: **PUA, Don** et al.; Western Digital Technologies, Inc., 3355 Michelson Drive, Suite 100, Irvine, California 92612 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

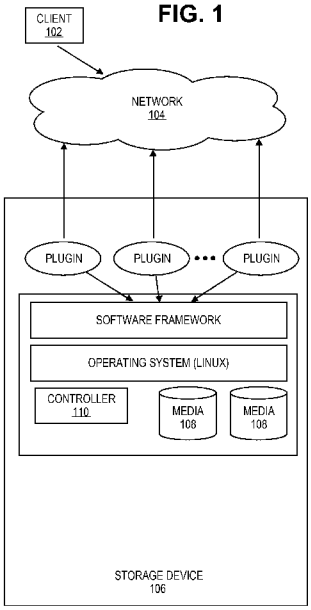
Published:

— *with international search report (Art. 21(3))*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

(54) Title: SOFTWARE FRAMEWORK FOR A STORAGE DEVICE

FIG. 1



(57) Abstract: The embodiments of present invention relates to methods and systems for a cross-platform framework for embedded systems. One embodiment provides a framework for a network attached storage or other storage devices. The framework provides a standardized structure for modules of software, such as plugins, to implement various features on the embedded system. In addition, the framework supports interoperability between software modules written in either native code, such as C, C++, and interpreted code, such as Java.

WO 2014/145132 A1

SOFTWARE FRAMEWORK FOR A STORAGE DEVICE

BACKGROUND

[0001] On embedded, and thus, resource-limited devices, updating to add new functionality and bug fixing becomes more and more necessary. Typically, a storage device, such as a network-attached storage (NAS), is constructed as a specialized computer or appliance. Normally, such devices use low-power hardware and having an embedded operating system, such as LINUX, that is configured specifically as a storage device.

[0002] Today, due to their increasing complexity, storage devices now run software to implement their various features and manage their interactions with other devices. Unfortunately, the software for each storage device is custom built and integrated into a single package with the operating system. This form of software development is difficult and often results in bugs, etc.

BRIEF DESCRIPTION OF THE DRAWINGS

[0003] Systems and methods which embody the various features of the invention will now be described with reference to the following drawings, in which:

[0004] FIG. 1 illustrates an exemplary data storage device that has been configured as a network attached storage system to illustrate an implementation of an embodiment of the present invention.

[0005] FIG. 1A illustrates an exemplary deployment of a framework in accordance with an embodiment of the present invention.

[0006] FIG. 2 illustrates an exemplary architecture of the framework in accordance with an embodiment of the present invention.

[0007] FIG. 3 illustrates exemplary structural elements of the framework in accordance with an embodiment of the present invention.

[0008] FIG. 4 illustrates an exemplary package.

[0009] FIG. 5 illustrates an exemplary structure of a manifest and its headers.

[0010] FIG. 6 illustrates an exemplary resolver used by the framework of one embodiment.

[0011] FIG. 7 illustrates an exemplary services object used by a framework of one embodiment of the present invention.

[0012] FIG. 8 illustrates an exemplary services observer used by a framework in one embodiment.

[0013] FIG. 9 illustrates an exemplary state chart of a framework in one embodiment.

[0014] FIG. 10 illustrates an exemplary resolver used in a framework of one embodiment.

[0015] FIG. 11 illustrates an exemplary bundle state chart.

[0016] FIG. 12 illustrates an exemplary services object used in a framework of one embodiment.

[0017] FIG. 13 illustrates an exemplary services observer object.

DETAILED DESCRIPTION

[0018] The current software architecture used in embedded devices makes it difficult for developers and vendors to write applications that interact in a consistent manner with other applications, for example, running on a storage device. It is also difficult for developers and vendors to write applications in different languages and update the program code. Accordingly it would be desirable to provide a different architecture for embedded devices, especially storage devices.

[0019] The embodiments relate to methods and systems for a cross-platform framework for embedded systems. The framework implements a plugin architecture that can be implemented on a variety of platforms, including embedded devices such as storage devices. The framework enables creation of components for applications in a modular fashion that can be assembled together and upgraded incrementally in whole or in part. One embodiment provides a framework for a network attached storage or other storage devices. The framework provides a standardized structure for modules of software, such as plugins, to implement various features on the embedded system. In some embodiments, the framework is based on native code that is compiled to run the embedded device, and thus, can support virtually any programming language used to write applications and plugins for the framework. In addition, the framework supports interoperability between software modules written in either native code, such as C++, and various clients. In some embodiments, the framework may support applications and plugins written in interpreted code, such as Java clients or plugins.

[0020] The framework provides an abstraction layer and makes code running on an embedded device, such as a storage device, modular and configurable. Thus, various embedded device products can be configurable with different features. In some embodiments, the framework also modifies which plugins are served.

[0021] A consistent programming model provided by the framework helps bundle developers cope with scalability issues in many different dimensions. The framework may run on a variety of devices whose differing hardware characteristics may affect many aspects of a service implementation. Consistent interfaces provided by the framework can ensure that the software components supported by the framework can be mixed and matched and still result in stable systems.

[0022] For example, the framework allows bundles to select an available implementation at run-time through the framework service registry. Bundles may register new services, receive notifications about the state of services, or look up existing

services to adapt to the current capabilities of the device. This aspect of the framework makes an installed bundle extensible after deployment: that is, new bundles can be installed for added features or existing bundles can be modified and updated without requiring the system to be restarted.

[0023] In one embodiment, the framework is implemented using native program code, such as C++. The native implementation of the framework provides several advantages. Dynamic loading of modules may be implemented with standard mechanisms. Dependencies between modules can be resolved when they are loaded. The framework can also provide significant advantages regarding performance and memory requirements. Some embodiments use open standards and execution of native binary code and are capable of running on any standard POSIX system.

[0024] The embodiments enable dynamic loading of code at runtime on the basis of shared object files and control the visibility of code modules (e.g. import/export-packages). The framework provides a mechanism to load different versions of the same package and, if desired, supports object-oriented code as well as different file formats, such as the Java Archive File Format (JAR) to package and deploy bundles. In some embodiments, the framework allows updates of bundles at runtime.

[0025] Furthermore, the embodiments provide functionality of a life-cycle layer (e.g. resolve/start/stop/update), a service layer (i.e. services, service listeners, filters), and support for various devices to participate in software-defined storage.

[0026] In one embodiment, the framework of the present invention provides an asynchronous, event-driven, message-passing, framework that implements a plugin architecture suitable for use across platforms and product lines. It enables the creation of components developed along the lines of a software factory that produces software parts engineered for assembly into products as well as upgrades to, and future versions of, existing products.

[0027] Certain embodiments of the inventions will now be described. These embodiments are presented by way of example only, and are not intended to limit the scope of the inventions. Indeed, the novel methods and systems described herein may be embodied in a variety of other forms. Furthermore, various omissions, substitutions and changes in the form of the methods and systems described herein may be made without departing from the spirit of the inventions. To illustrate some of the embodiments, reference will now be made to the figures.

[0028] FIG. 1 shows a system 100 in which the embodiments may be implemented. For purposes of illustration, an exemplary framework is shown implemented on a network attached storage (NAS). However, those skilled in the art will recognize that any type of embedded device or device capable of running a framework are within the principles of the present invention.

[0029] As shown, the system 100 may comprise a client or host device 102, a network 104, and a storage device 106. The storage device 106 may be implemented as a NAS device having a plurality of storage media 108. These components will now be further described.

[0030] Host device 102 refers to any computing device used to access files shared by storage device 106. For example, the host device 102 may be a personal computer, a laptop computer, a tablet computer, a mobile phone, etc. Such devices are well known to those skilled in the art.

[0031] Network 104 provides a communications infrastructure for communications between the host device 102 and the storage device 106. Network 104 may comprise various components and network elements, such as routers, switches, hubs, etc.

[0032] The network 104 may support various communications protocols, such as Ethernet, Internet protocols, etc. In addition, the network 104 may be implemented as a wired or wireless network or a combination thereof.

[0033] Storage device 106 refers to any data storage device capable of providing or sharing access to a file. Storage device 106 may be implemented as a hard disk drive, a solid-state drive, a hybrid drive, a direct attached storage device, etc.

[0034] In the present disclosure, by way of example, storage device 106 is implemented as a NAS device. As a NAS device, storage device 106 provides file level access and shares files with host device 102 over network 104. The storage device 106 may be implemented using known hardware, firmware, and software. For example, as a NAS device, the storage device 106 may be implemented as a specialized computer or appliance configured to store and serve files over the network 104. In some embodiments, the storage device 106 comprises a processor, such as an ARM, PowerPC, or MIPS processor.

[0035] Storage media 108 refers to the medium on which the storage device 106 stores data. For example, storage media 108 may comprise disks or magnetic media, solid-state memory, optical media, etc. As shown, the storage device 106 may comprise

multiple storage media 108 for purposes of capacity and redundancy. For example, as a NAS device, the storage device 106 may support various RAID levels.

[0036] Controller 110 represents the hardware and software that controls the operations of the storage device. In some embodiments, the controller 110 may also comprise one or more other components to supplement its operations, such as an on-chip RAID controller, a memory or disk cache, etc. As will be described further below, the controller 110 may execute an operating system 112 and a framework 114.

[0037] Operating system 112 represents the software executed by the controller 110 to manage the hardware resources and provide various services, such as web services, for applications running on the storage device 106. The controller 110 may support a variety of operating systems, such as UNIX, LINUX, Windows, and the like, which are known to those skilled in the art. For example, in one embodiment, the controller 110 may run based on an operating system, such as Plan 9, UNIX, LINUX, and the like.

[0038] In addition, in one embodiment, the storage device 106 comprises a framework 114 provides a standardized structure for modules of software, such as plugins, to implement various features on the storage device 106. In some embodiments, the framework is implemented with native code, such as C++, and thus, can support virtually any programming language. In addition, the framework supports interoperability between software modules written in either native code, such as C++, and interpreted code, such as Java. For example, in one embodiment, a Java Runtime Environment ("JRE") may be installed on the storage device 106 and execute on the framework 114 to support Java-based software and plugins.

[0039] The embodiments provide a framework 114 that can run on any standard POSIX-compliant operating system, such as LINUX. The embodiments use shared objects to provide software modularity. For example, each package (i.e., C++ namespace) can be represented by a shared object file in the Executable and Linking Format or Executable and Linkable Format ("ELF"). This allows implementing the dynamic import and export of packages between bundles at runtime with standard dlopen and ELF mechanisms. In some embodiments, the framework 114 may support other types of file formats, such as Common Object File Format ("COFF"), Win32, Object Module Format ("OMF"), etc.

[0040] Bundles can be implemented as ZIP-archives containing a standardized manifest and shared object files representing packages. Without any imports in the manifest, shared objects of different bundles may be isolated from each other.

[0041] To establish visibility between shared objects, the runtime system of the framework 114 adds ELF dependencies to the shared object files after resolving available packages according to specied imports. Thus, in some embodiments, codesharing functionality is provided by standard ELF mechanisms, which are highly mature.

[0042] Moreover, the embodiments of the framework 114 provide life-cycle layer functionality (e.g., starting, updating and stopping of bundles). For example, the framework 114 may implement a service layer with a service registry including support for RFC1960-compliant filters. Finally, the embodiments support an event and listener concept by offering framework, bundle and service listeners.

[0043] In one embodiment, the framework 114 employs a message passing interface. For example, the framework 114 may employ the BOOST.MPI library. The BOOST.MPI may serve as a wrapper for the message passing interface (MPI). The Boost.MPI is a library for message passing in high-performance parallel applications. A Boost.MPI program is one or more processes that can communicate either via sending and receiving individual messages (point-to-point communication) or by coordinating as a group (collective communication). Unlike communication in threaded environments or using a shared-memory library, Boost.MPI processes can be spread across many different machines, possibly with different operating systems and underlying architectures.

[0044] Boost.MPI is a C++-friendly interface to the standard Message Passing Interface (MPI), the most popular library interface for high-performance, distributed computing. MPI defines a library interface, available from C, Fortran, and C++, for which there are many MPI implementations.

[0045] The Boost.MPI library provides an alternative C++ interface to MPI that better supports modern C++ development styles, including complete support for user-defined data types and C++ Standard Library types, arbitrary function objects for collective algorithms, and the use of modern C++ library techniques to maintain maximal efficiency.

[0046] In one embodiment, the Boost.MPI is configured to support the functionality in MPI 1.1. The abstractions in Boost.MPI allow calls to the underlying C MPI library.

[0047] For example, in one embodiment, the Boost.MPI is used to support:

Communicators: Boost.MPI supports the creation, destruction, cloning, and splitting of MPI communicators, along with manipulation of process groups.

Point-to-point communication: Boost.MPI supports point-to-point communication of primitive and user-defined data types with send and receive operations, with blocking and non-blocking interfaces.

Collective communication: Boost.MPI supports collective operations such as reduce and gather with both built-in and user-defined data types and function objects.

MPI Datatypes: Boost.MPI can build MPI data types for user-defined types using, for example, the Boost.Serialization library.

Separating structure from content: Boost.MPI can transfer the shape (or "skeleton") of complex data structures (lists, maps, etc.) and then separately transfer their content. This facility optimizes for cases where the data within a large, static data structure needs to be transmitted many times.

[0048] In the embodiments, Boost.MPI can be accessed either through its native C++ bindings, or through an alternative, such as a Python interface. In another embodiment, the framework 114 may utilize a communications protocol, such as 9P provided by a Plan 9 operating system.

[0049] The framework 114 provides a general-purpose, secure, and managed framework that supports the deployment of extensible and down-loadable applications known as bundles. The framework 114 manages the installation and update of bundles in a dynamic and scalable fashion. To achieve this, it manages the dependencies between bundles and services in detail. It provides the bundle developer with the resources necessary to take advantage of the platform independence and dynamic code-loading capability in order to easily develop services for small-memory devices that can be deployed on a large scale.

[0050] In one embodiment, the framework 114 enables the storage device 106 to support and implement software-defined storage. In particular, the framework 114 provides a platform by which the storage device 106 may virtualize its storage capacity and a communications protocol by which to establish virtualized storage on the storage

device 106 alone or in cooperation with other storage devices (not shown). For example, the framework 114 may serve as a platform running on the storage device 106 to deliver block and file storage services. In one embodiment, the framework 114 provides an abstraction layer for the underlying hardware of the storage device 106. Furthermore, the framework 114 may provide one or more APIs for various storage management functions, such as volume management, storage provisioning services, RAID configuration, etc.

[0051] FIG. 1A illustrates an exemplary deployment implementation of the framework 114. The framework 114 uses bundles to deploy one or more services. Bundles contain the libraries, resources and additionally a manifest file describing the content of the bundle. In C, libraries contain only the Objects compiled out of the source. In the prior art, it is not possible to add resources and the manifest. Accordingly, in the embodiments, a zip file is used in which the library and resources are packed.

[0052] FIG. 2 illustrates an exemplary architecture of the framework 114. As shown, the framework 114 may comprise the following layers: a bundles layer 200, an execution environment layer 202, a module layer 204, a life cycle layer 206, a service layer 208, and a security layer 210. These components of the framework 114 will now be further described.

[0053] The bundles in bundle layer 200 provide the program code for the plugins requesting various services of the framework 114 and other plugins. In the embodiments, the framework 114 defines a unit of modularization, called a bundle. A bundle is comprised of binaries and other resources, which together can provide functions to end users. Bundles can share bundle capabilities and bundle requirements in a well-defined way. Thus, in the framework 114, bundles are the entities for deploying applications.

[0054] As noted, a bundle may be deployed as a ZIP file. ZIP files are used to store applications and their resources in a standard ZIP-based file format. A bundle is a ZIP file that contains the resources necessary to provide some functionality. These resources may be HTML files, help files, icons, etc.

[0055] The bundle may also contain a manifest file describing the contents of the ZIP file and providing information about the bundle. This file uses headers to specify information that the framework 114 uses to correctly activate a bundle. For example, it states dependencies on other resources that must be available to the bundle before it can run.

[0056] Once a bundle is started, its functionality is provided and services are exported to other bundles installed in the framework 114. The bundle may comprise various components to determine how the bundle executes. For example, a bundle-activation policy specifies how the framework should activate the bundle once started. Some embodiments of possible headers are described in the following descriptions below. A bundle-activator header specifies the name of the class used to start and stop the bundle. A bundle header may hold a comma-separated list of category names.

[0057] Another header defines a comma-separated list of ZIP file path names or directories inside the bundle containing classes and resources. A version header defines that the bundle has been released according to a serialized version. A native code header contains a specification of native code libraries contained in this bundle.

[0058] A symbolic name header specifies a non-localizable name for this bundle. The bundle symbolic name together with a version that identifies a unique bundle though it can be installed multiple times in a framework.

[0059] An update header specifies a URL where an update for this bundle is available, for example, in the form of a ZIP file. A dynamic import-package header contains a comma-separated list of package names that should be dynamically imported when needed.

[0060] An export-Package header contains a declaration of exported packages. A require-bundle header specifies that all exported packages from another bundle must be imported, effectively requiring the public interface of another bundle. A require-capability header specifies that a bundle requires other bundles to provide a capability.

[0061] In one embodiment, the execution environment layer 202 establishes the execution environment for running the bundles 200.

[0062] In one embodiment, the module layer 204 defines the modularization model. It may have strict rules for sharing between bundles or hiding packages from other bundles. In the module layer 204, since bundles depend on one or more external entities. A bundle can require other bundles, an execution environment, a specific window system, a DLL or shared library, an extender, a specific peripheral, etc. Once a bundle starts, it assumes that those dependencies are satisfied. Bundles can express their external dependencies via requirements on capabilities that are provided by the run- time environment or other bundles. For example, BOOST provides an easy to use interface, such as MPI 1.1 and provides the ability to use any version.

[0063] Capabilities are attribute sets in a specific name space and requirements are filter expressions that assert the attributes of the capabilities of their corresponding name space. A requirement is satisfied when there is at least one capability that matches the filter. A name space also provides the semantics of matching a requirement to a capability. Bundles can only provide their capabilities when their requirements are satisfied, that is, requirements are transitive.

[0064] In one embodiment, the life cycle layer 206 provides a life cycle interface, such as an application programming interface (API) to bundles such as the bundles in bundle layer 200. This API provides a runtime model for bundles. It defines how bundles are started and stopped as well as how bundles are installed, updated and uninstalled. Additionally, it provides a comprehensive event API to allow a management bundle to control the operations of the service platform.

[0065] In one embodiment, the service layer 208 provides a dynamic, concise and consistent programming model for bundle developers, simplifying the development and deployment of service bundles by de-coupling the API used the service's specification from its implementations. The embodiments may support any variety of services, including web services, web application servers, web servers, etc. This model allows bundle developers to bind to services only using their interface specifications. The selection of a specific implementation, optimized for a specific need or from a specific vendor, can thus be deferred to run-time.

[0066] In the service layer 208, bundles are built around a set of cooperating services available from a shared service registry. Such services are defined semantically by its service interface and implemented as a service object. The service interface can be specified with as few implementation details as possible.

[0067] The service object is owned by, and runs within, a bundle. In one embodiment, this bundle must register the service object with the framework service registry so that the service's functionality is available to other bundles under control of the framework.

[0068] In one embodiment, the framework 114 manages dependencies between the bundle owning the service and the bundles using it. For example, when a bundle is stopped, all the services registered with the framework 114 by that bundle must be automatically unregistered. The framework 114 maps services to their underlying service objects, and provides a simple but powerful query mechanism that enables a bundle to request the services it needs. The framework 114 also provides an event

mechanism so that bundles can receive events of services that are registered, modified, or unregistered.

[0069] In one embodiment, the security layer 210 defines a secure packaging format as well as the runtime interaction with the framework 114. In one embodiment, the security layer 210 is based on the X.509 v3 digital certificate model. A manifest for approved bundles contains the minimum required capabilities and authorizations. It may be digitally signed using a private key and is authenticated with the public key resident in device firmware. Digitally signing is a security feature that verifies the signer and ensures that the content has not been modified after it was signed by the principal. In one embodiment, the digital signing is based on public key cryptography. In one embodiment, the plugins running on the framework 114 are signed by a trusted entity, such as the manufacturer of the storage device 106. One or more of a variety of entities may sign the plugins to enhance the security and/or authentication of the plugins. In addition, a public key of one or more entities may be persistently stored on the storage device 106. For example, a public key may be burned into firmware of the controller 110. In one embodiment, an intermediate certificate installed on the storage device 106 comprises various metadata.

[0070] FIG. 3 illustrates exemplary structural elements of the framework in accordance with an embodiment of the present invention. The Framework forms the core of a general-purpose, secure and managed framework that supports the deployment of extensible and downloadable applications known as bundles. It may be implemented as a singleton and use a pointer to implementation (PIMPL) idiom to separate the implementation from the framework API description itself. Below are some exemplary descriptions of the structural elements of the framework.

[0071] Exemplary Framework API

std::vector< std::shared_ptr< bundle_t >> get_bundles(): Returns a vector of shared_ptr to all of the bundles currently installed in the Framework;

std::shared_ptr< bundle_t > get_bundle(bundle_id_t): Returns a shared pointer to the bundle object specified by the unique identifier.

service_registration_t register_service

service_reference_t get_service_reference(std::string)

[0072] FIG. 4 illustrates an exemplary package of bundles. Bundles are entities that are visible in normal application programming. An example is provided below.

bundle.std::shared_ptr< bundle_t > get_bundle(): Returns a shared pointer to the bundle object.

bundle_identifier_t get_bundle_id()

std::string get_symbolic_name()

std::string get_vendor()

bundle_version_t get_version()

Manifests

Each bundle object contains a manifest object that provides access to the contents of the manifest file delivered as part of the bundle.

Exemplary Manifest API

const ::string & bundle_copyright().

const std::string& bundle_contact_address().

const std::string& bundle_license().

void parse_manifest(const std::string& path).

void parse_manifest(boost::filesystem::filebuf infile).

const std::string& bundle_name().

const std::string& symbolic_name().

const ::string& bundle_vendor().

manifest_version_t manifest_version().

std::vector< std::string > require_bundle(): Returns a list of required bundle symbolic-names with optional attribute assertions.

std::vector< std::string > provide_capability(): Specifies that a bundle provides a set of capabilities in the form of one or more named interfaces.

std::vector< std::string > require_capability(): Specifies that a bundle requires a set of capabilities in the form of one or more named interfaces.

[0073] FIG. 5 illustrates an exemplary structure of a manifest and its headers that may be used by the framework 114. A plugin bundle carries descriptive information about itself in the manifest file that is contained in the bundle archive, for example, under the name META-INF/MANIFEST.MF. In one embodiment, specified manifest headers are listed in the following sections.

Manifest-Version	
Bundle-Activator	

Bundle-ContactAddress	
Bundle-Copyright	
Bundle-License	
Bundle-ManifestVersion	
Bundle-Name	
Bundle-SymbolicName	
Bundle-Vendor	
Bundle-Version	
Provide-Capability	
Require_Capability	
Require-Bundle	
Bundle-LinixCapabilities	The Bundle-Capabilities specifies which Linux capabilities will be granted to the plugin in the bundle. Bundle-LinuxCapabilities := operation(';' operation)* Operation := CAP_AUDIT_CONTROL; CAP_AUDIT_WRITE; CAP_CHOWN; CAP_DAC_OVERRIDE; CAP_DAC_READ_SEARCH; CAP_FOWNER; CAP_FSETID; CAP_IPC_LOCK; CAP_IPC_OWNER; CAP_KILL; CAP_LEASE; CAP_LINUX_IMMUTABLE; CAP_MAC_ADMIN; CAP_MAC_OVERRIDE; CAP_MKNOD; CAP_NET_ADMIN;CAP_NET_BIND_SERVICE; CAP_NET_ADMIN; CAP_NET_BIND_SERVICE; CAP_NET_BROADCAST; CAP_NET_RAW; CAP_SETGID; CAP_SETPCAP; CAP_SETUID; CAP_SYS_ADMIN;

	CAP_SYS_BOOT; CAP_SYS_CHROOT; CAP_SYS_MODULE; CAP_SYS_NICE; CAP_SYS_PACCT; CAP_SYS_PTRACE; CAP_SYS_RAWIO; CAP_SYS_RESOURCE; CAP_SYS_TIME; CAP_SYS_TTY_CONFIG In one embodiment, the plugin is an executable, the Linux Capabilities may be applied to the process thread, otherwise to the file or files in combination with access control lists ("ACLs").

[0074] Manifest Header Factory

[0075] In one embodiment, the Manifest Header Factory class is implemented as a singleton. Its purpose is to instantiate specific manifest_header objects based on the manifest header name. This class enables polymorphic dispatch and, if new header types are required, they can be added without changing any existing code. An example is provided below.

Exemplary Manifest Header Factory API

static manifest_header_factory& instance()

void register_header()

Of note, every manifest header handler registers a function

void unregister_header()

[0076] FIG. 6 illustrates an exemplary resolver used by the framework of one embodiment. The bundle resolver uses the bundle_requirement and bundle_capability information contained in the manifest headers to identify the 'best-fit' for any given bundle. In the event of 'unresolvable' conflicts, the resolver will create a renamed version of 'best' candidate, resolve it, and modify the original bundle dependencies in the elf to reflect the 'new' library name. The modified version of the bundle will then be loaded and destroyed on framework exit.

[0077] FIG. 7 illustrates an exemplary services object used by a framework of one embodiment of the present invention. The service object is owned by, and runs within a bundle. This bundle must register the service object with the Framework service registry so that the service's functionality is available to other bundles under control of the Framework.Services API. An example is provided below.

std::shared_ptr<?> get_service(shared_ptr<service_reference>)

where Service Interfaces: is a service interface is the specification of the service's public methods.

[0078] FIG. 8 illustrates an exemplary services observer used by a framework in one embodiment.

[0079] FIG. 9 illustrates an exemplary statechart of a framework in one embodiment. Upon exiting the INSTALLED state the following actions may occur:

Event handling is enabled,

The security manager is configured,

Any installed bundle is in the INSTALLED state, and

Framework services are available,

[0080] Resolved State

[0081] When the framework is in the RESOLVED state, it is operational but none of its bundles are active. As long as the framework is in this state, new bundles can be installed without any installed code interfering. Existing bundles must all be in the INSTALLED state. The framework_start event is required to move the framework to the STARTING state.

[0082] Alternatively, there may be one state where bundles are resolved and another where (during shutdown) resources previously constructed are destroyed before transitioning bask to the framework INSTALLED state.

[0083] FIG. 10 illustrates an exemplary resolver used in a framework of one embodiment.

[0084] Starting

[0085] In the STARTING state, the framework starts all the installed bundles. Once all the bundles have been started, the framework transitions to the ACTIVE state.

[0086] Active

[0087] In the framework ACTIVE state, all installed bundles previously recorded as being started must be in the bundle ACTIVE state.

[0088] Stopping

[0089] When the framework is shutdown, if first enters the framework STOPPING state where all bundles in the ACTIVE state are stopped. The framework then transitions to the RESOLVED state where resources are destroyed.

[0090] FIG. 11 illustrates an exemplary bundle state chart.

[0091] Installed

[0092] The bundle has been successfully installed. When a bundle is installed, it is stored in the persistent storage of the Framework and remains there until it is explicitly uninstalled. Whether a bundle has been started or stopped must be recorded in the persistent storage of the Framework. A bundle that has been persistently recorded as started must be started whenever the Framework starts until the bundle is explicitly stopped.

[0093] Resolved

[0094] All classes that the bundle needs are available. This state indicates that the bundle is either ready to be started or has stopped.

[0095] Starting

[0096] The bundle is being started. The bundle_activator.start method will have been called but has not yet returned.

[0097] Active

[0098] The bundle has been successfully activated and is running. Its bundle_activator.start method has been called and has returned.

[0099] Stopping

[00100] The bundle is being stopped. The bundle_activator.stop method has been called but the stop method has not yet returned.

[00101] Uninstalled

[00102] The bundle has been uninstalled. It cannot move into another state.

[00103] FIG. 12 illustrates an exemplary services object used in a framework of one embodiment. A service may be defined semantically by its service interface[s] and implemented as a service object. Service interfaces are named and versioned and composed of operations.

[00104] The service object is owned by, and runs within, a bundle. This bundle must register the service object with the framework service registry so that the service's functionality is available to other bundles under control of the Framework.

[00105] Dependencies between the bundle owning the service and the bundles using it are managed by the framework. The framework maps services to their underlying service objects using the resolver. The framework also provides an event mechanism so that bundles can receive events from services that are registered. FIG. 13 illustrates an exemplary services observer object.

[00106] The features and attributes of the specific embodiments disclosed above may be combined in different ways to form additional embodiments, all of which fall

within the scope of the present disclosure. Although the present disclosure provides certain embodiments and applications, other embodiments that are apparent to those of ordinary skill in the art, including embodiments, which do not provide all of the features and advantages set forth herein, are also within the scope of this disclosure. Accordingly, the scope of the present disclosure is intended to be defined only by reference to the appended claims.

WHAT IS CLAIMED IS:

1. An embedded device implementing a framework, said embedded device comprising:

a hardware platform comprising a set of hardware components and configured to execute program code based on a processor architecture;

an operating system running on the hardware platform;

a framework application running on the operating system and configured to control execution of modules of software written in program code native to the hardware platform and modules of software written in interpreted code, wherein the framework application is configured to provide interoperability between the modules of software written in native program code and the modules of software written in native program code; and

a set of modules running on the framework and interfacing the framework application via an application programming interface.

2. The embedded device of claim 1, wherein the hardware platform comprises an embedded processor.

3. The embedded device of claim 1, wherein the hardware platform is a network attached storage device.

4. The embedded device of claim 1, wherein the operating system is a Linux operating system.

5. The embedded device of claim 1, wherein the operating system is a UNIX-based operating system.

6. The embedded device of claim 5, wherein the operating system is a Plan 9 operating system.

7. The embedded device of claim 1, wherein the operating system is a Windows operating system.

8. The embedded device of claim 1, wherein the framework is configured to control execution of software modules written in C.

9. The embedded device of claim 1, wherein the framework is configured to control execution of software modules written in C++.

10. The embedded device of claim 1, wherein the framework provides a set of native code libraries that can be loaded by software modules written in a programming language that is native to the hardware platform.

11. The embedded device of claim 1, wherein the set of modules comprises a plug-in.

12. A method of controlling execution of program code on an embedded device via a framework, said method comprising:

receiving software modules on to the embedded device, wherein the software modules may be written in an interpreted program code or a program code native to a hardware platform of the embedded device;

authenticating by the framework the software modules;

determining a set of permissions to the software modules based on the authentication;

verifying dependencies of the software modules needed for execution of the software modules;

compiling the software modules into program code that is executable on the hardware platform of the embedded device; and

controlling, by the framework, execution of the software modules, wherein the framework provides interoperability between software modules written in interpreted program code and software modules written in native program code.

13. The method of claim 12, wherein the embedded device is a network attached storage.

14. The method of claim 12, wherein receiving software modules comprises receiving software modules into a directory indicated by a pointer configured in the framework.

15. The method of claim 12, wherein authenticating the software modules comprises authenticating the software modules based on a digital signature.

16. The method of claim 12, wherein determining the set of permissions comprises determining the set of permissions based on a digital certificate indicated by a manifest for the software modules.

17. The method of claim 12, wherein determining dependencies of the software modules comprises determining a dependency on a native program code library based on a header provided in the software modules and a manifest for the software modules.

18. The method of claim 12, wherein controlling execution of the software modules comprises controlling execution of a software module written in native code.

19. The method of claim 12, wherein controlling execution of the software modules comprises controlling execution of a software module written in C++.

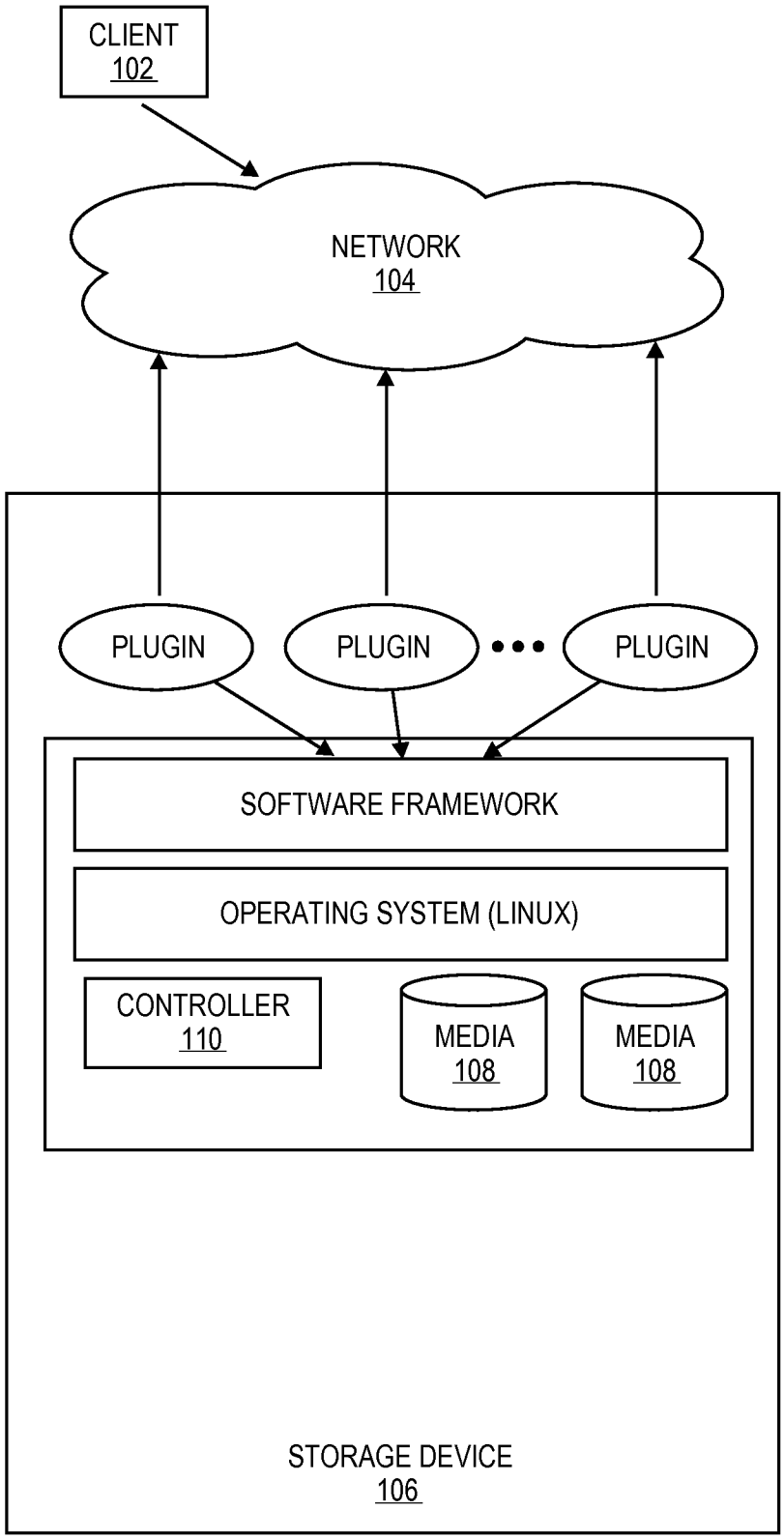

CLIENT
102
NETWORK
104
PLUGIN
PLUGIN
PLUGIN
SOFTWARE FRAMEWORK
OPERATING SYSTEM (LINUX)
CONTROLLER
110
MEDIA
108
MEDIA
108
STORAGE DEVICE
106

FIG. 1

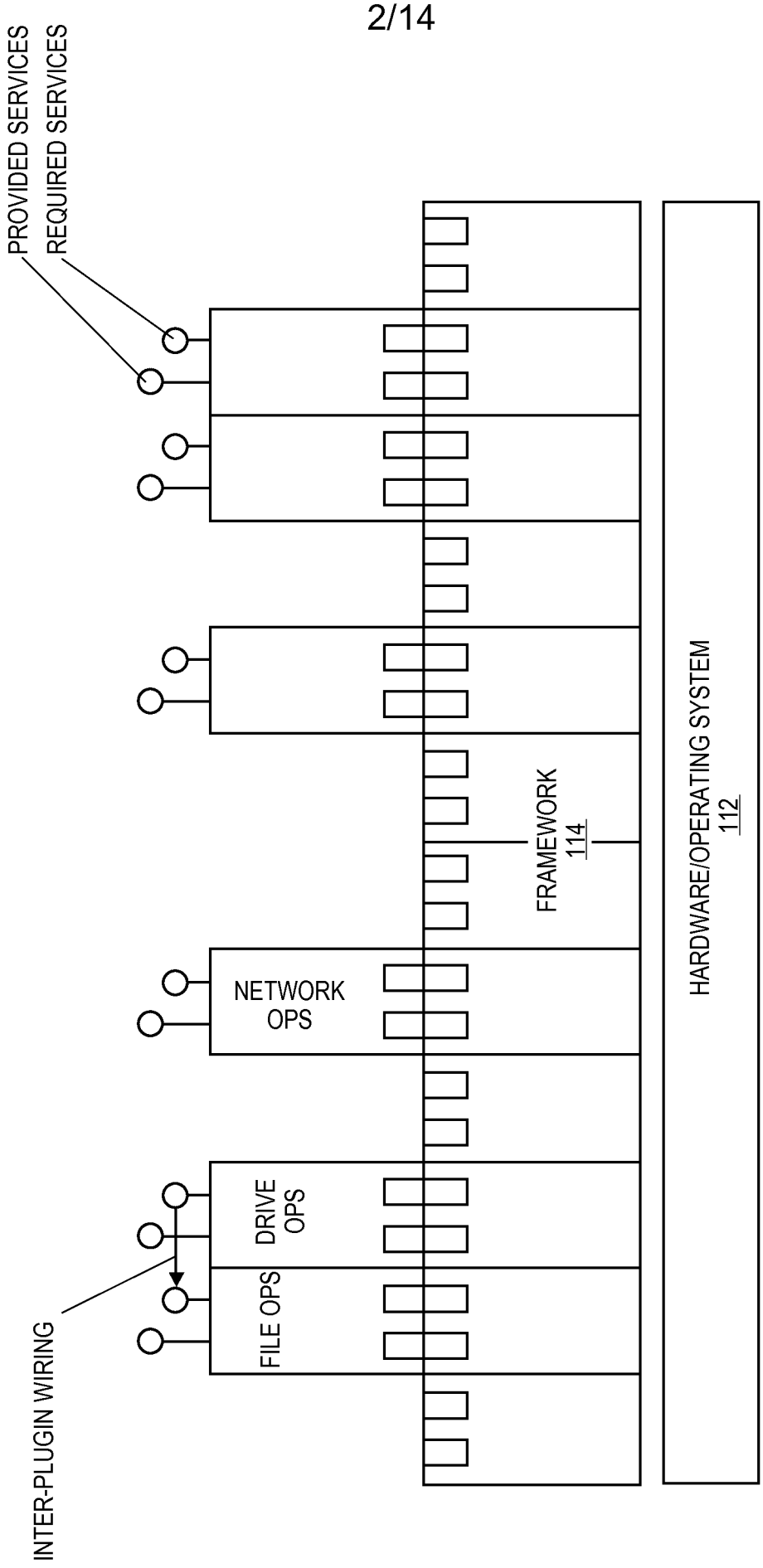

PROVIDED SERVICES
REQUIRED SERVICES
FRAMEWORK
114
NETWORK
OPS
DRIVE
OPS
FILE OPS
INTER-PLUGIN WIRING
HARDWARE/OPERATING SYSTEM
112

FIG. 1A

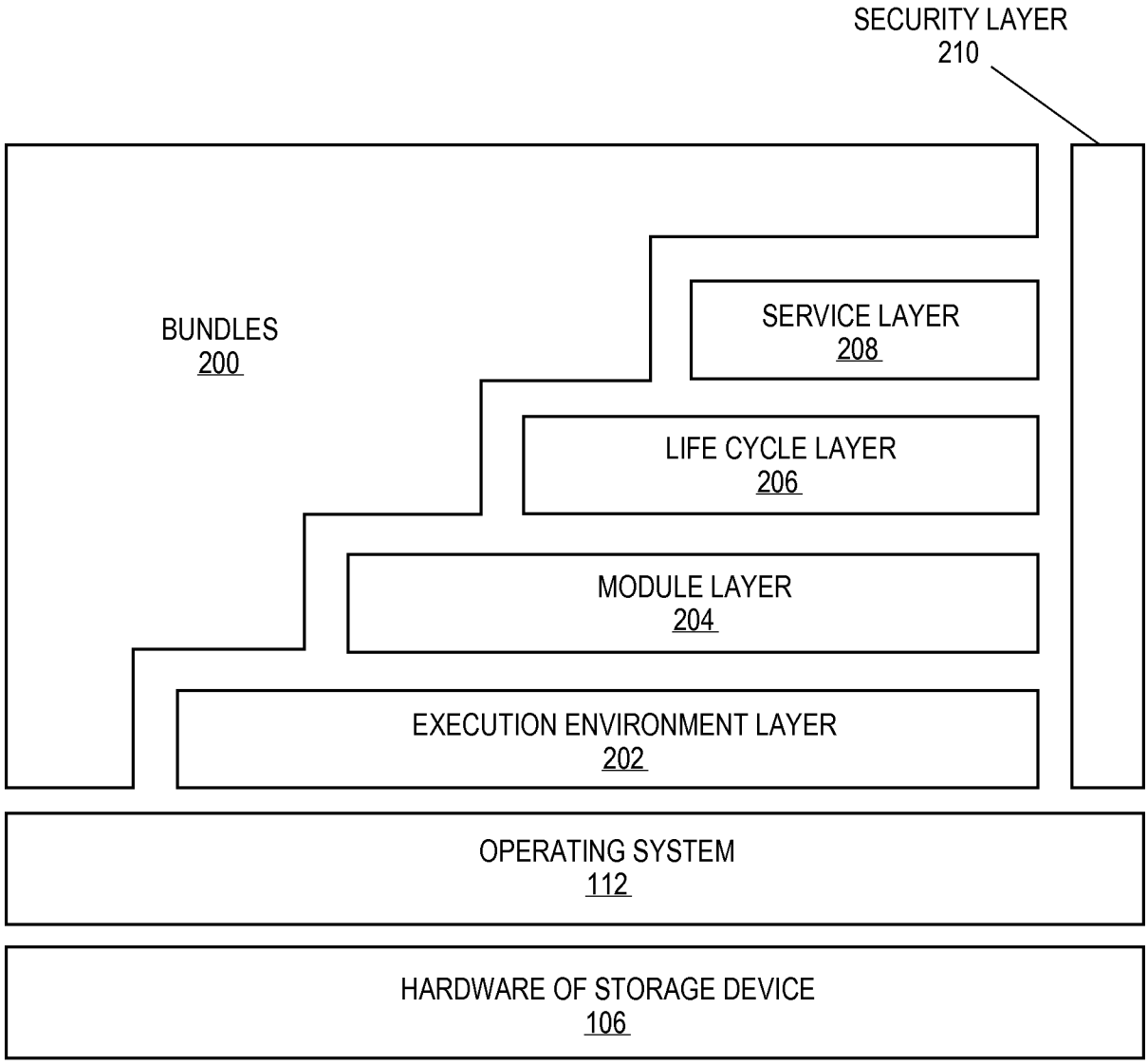

SECURITY LAYER
210
BUNDLES
200
SERVICE LAYER
208
LIFE CYCLE LAYER
206
MODULE LAYER
204
EXECUTION ENVIRONMENT LAYER
202
OPERATING SYSTEM
112
HARDWARE OF STORAGE DEVICE
106

FIG. 2

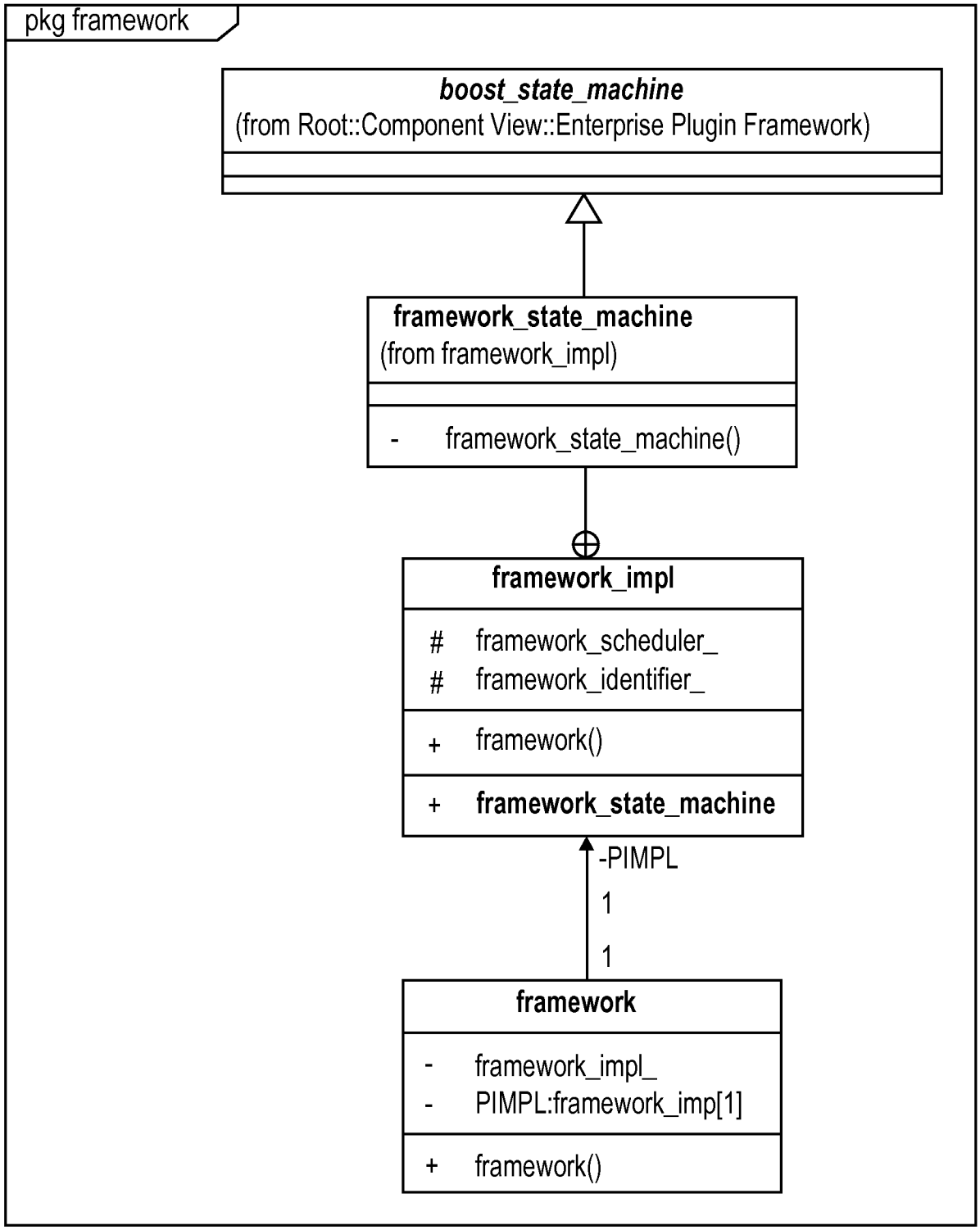

pkg framework
boost_state_machine
(from Root::Component View::Enterprise Plugin Framework)
framework_state_machine
(from framework_impl)
- framework_state_machine()
framework_impl
framework_scheduler_
framework_identifier_
+ framework()
+ framework_state_machine
-PIMPL
1
1
framework
- framework_impl_
- PIMPL:framework_imp[1]
+ framework()

FIG. 3

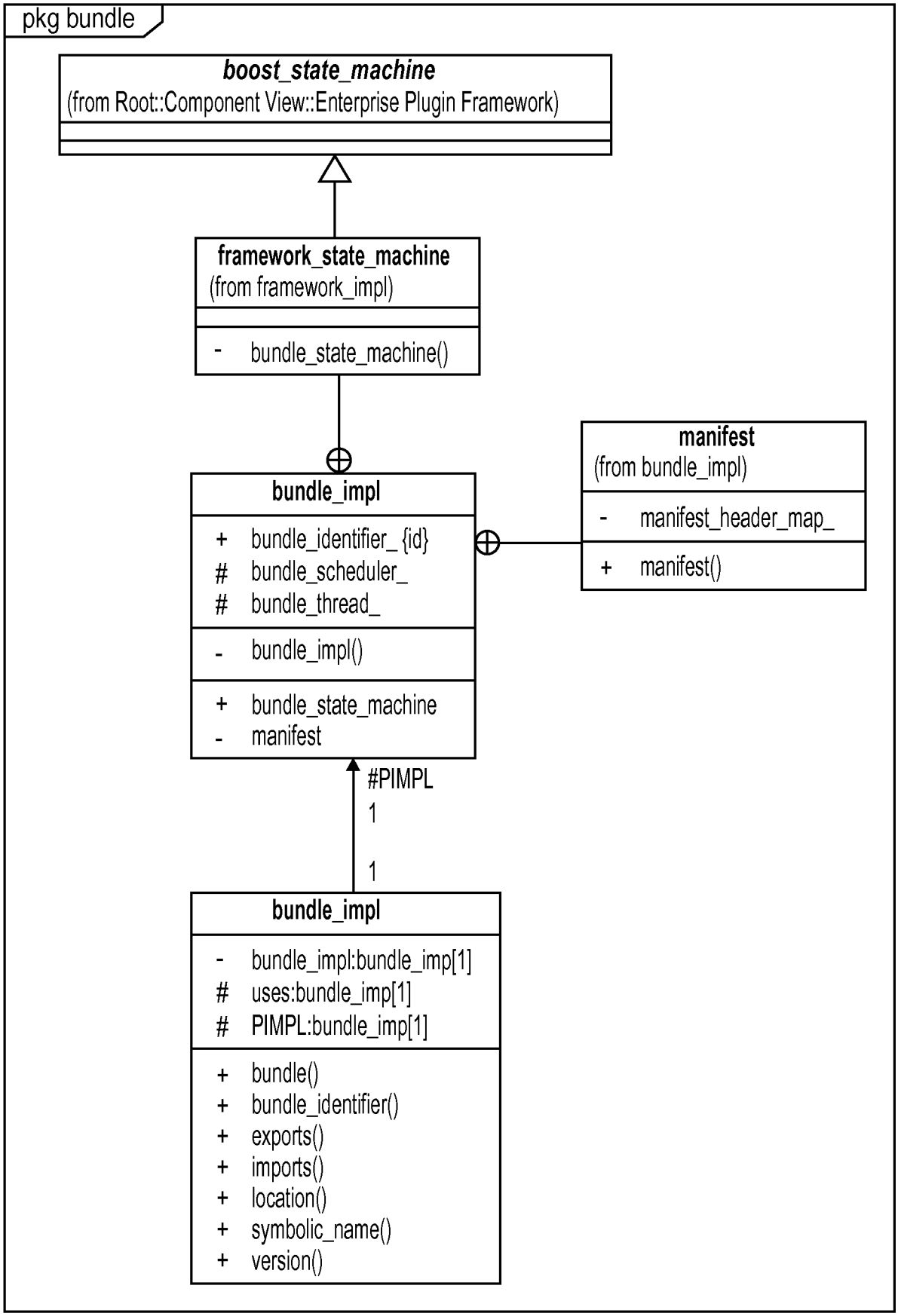

pkg bundle
boost_state_machine
(from Root::Component View::Enterprise Plugin Framework)
framework_state_machine
(from framework_impl)
- bundle_state_machine()
manifest
(from bundle_impl)
- manifest_header_map_
+ manifest()
bundle_impl
+ bundle_identifier_ {id}
# bundle_scheduler_
# bundle_thread_
- bundle_impl()
+ bundle_state_machine
- manifest
#PIMPL
1
1
bundle_impl
- bundle_impl:bundle_imp[1]
uses:bundle_imp[1]
PIMPL:bundle_imp[1]
+ bundle()
+ bundle_identifier()
+ exports()
+ imports()
+ location()
+ symbolic_name()
+ version()

FIG. 4

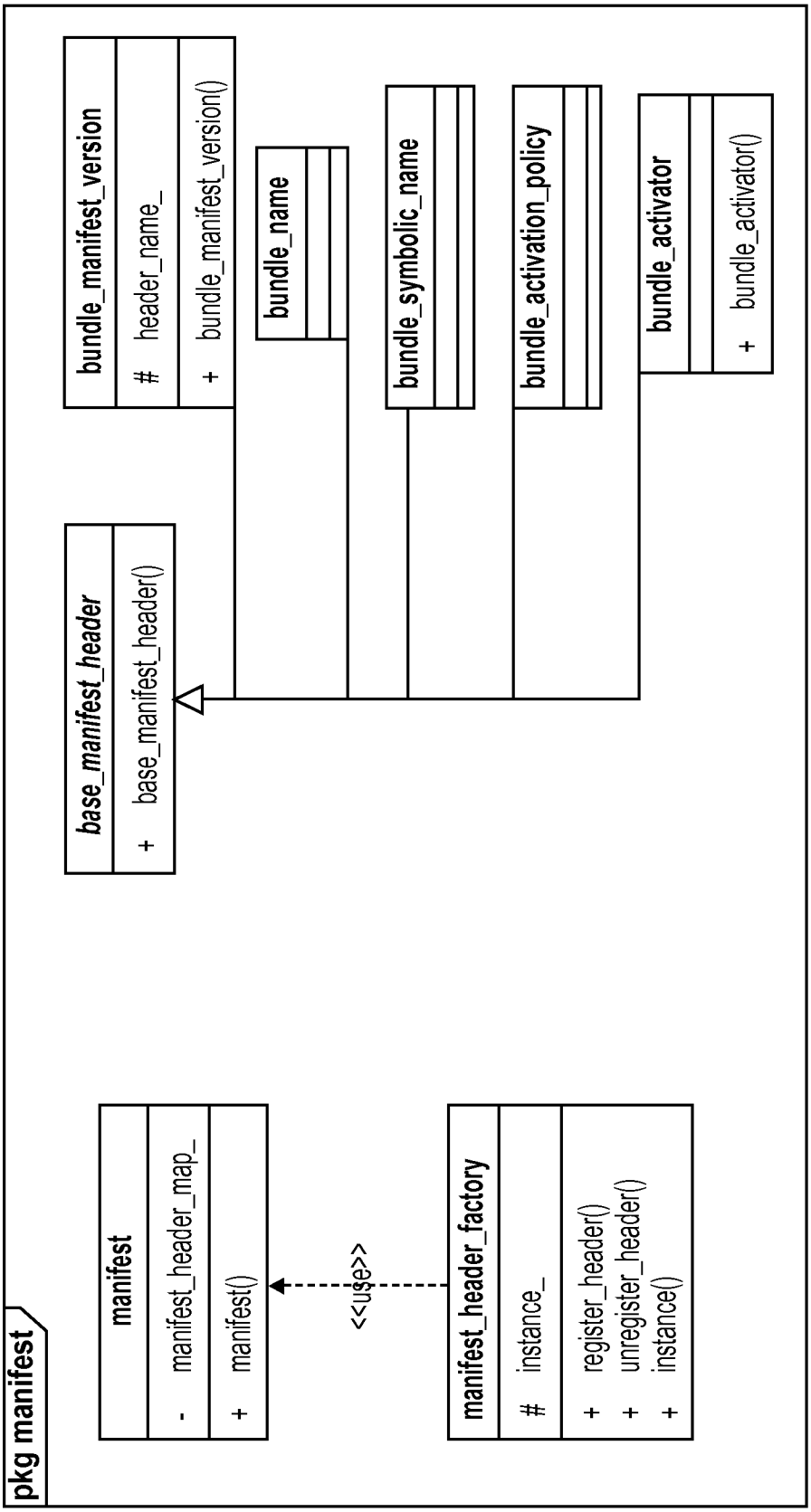

pkg manifest
manifest
- manifest_header_map_
+ manifest()
<<use>>
manifest_header_factory
instance_
+ register_header()
+ unregister_header()
+ instance()
base_manifest_header
+ base_manifest_header()
bundle_manifest_version
header_name_
+ bundle_manifest_version()
bundle_name
bundle_symbolic_name
bundle_activation_policy
bundle_activator
+ bundle_activator()

FIG. 5

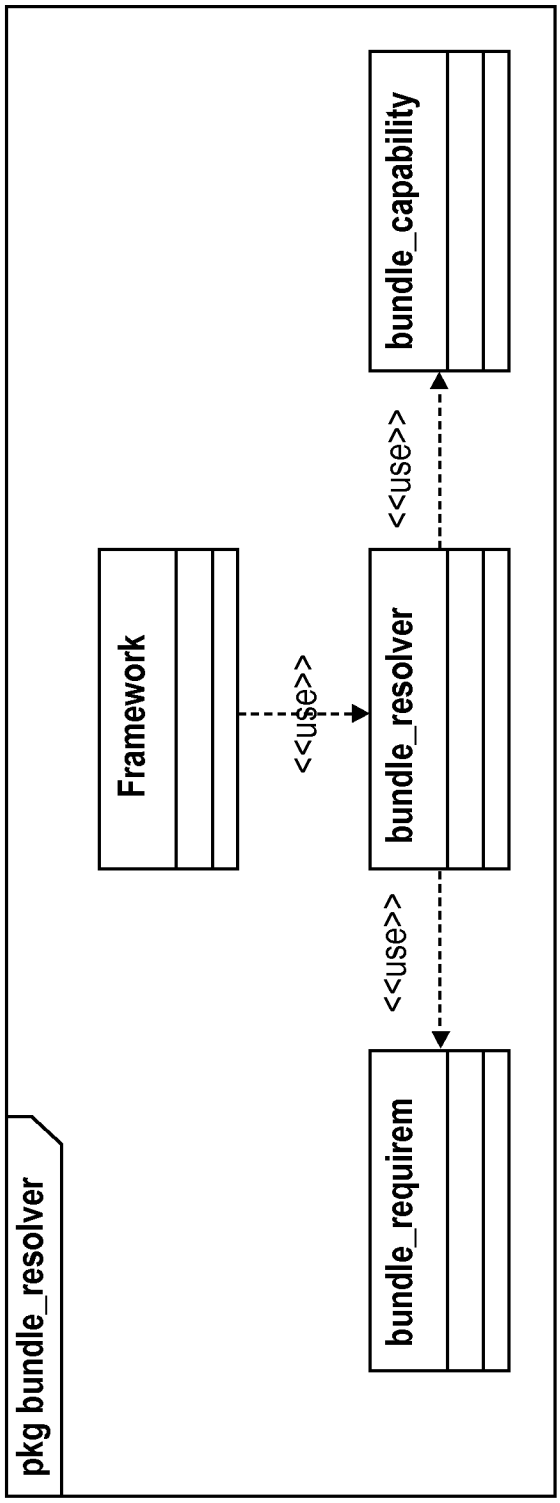

pkg bundle_resolver
Framework
<<use>>
bundle_resolver
<<use>>
bundle_capability
<<use>>
bundle_requirem

FIG. 6

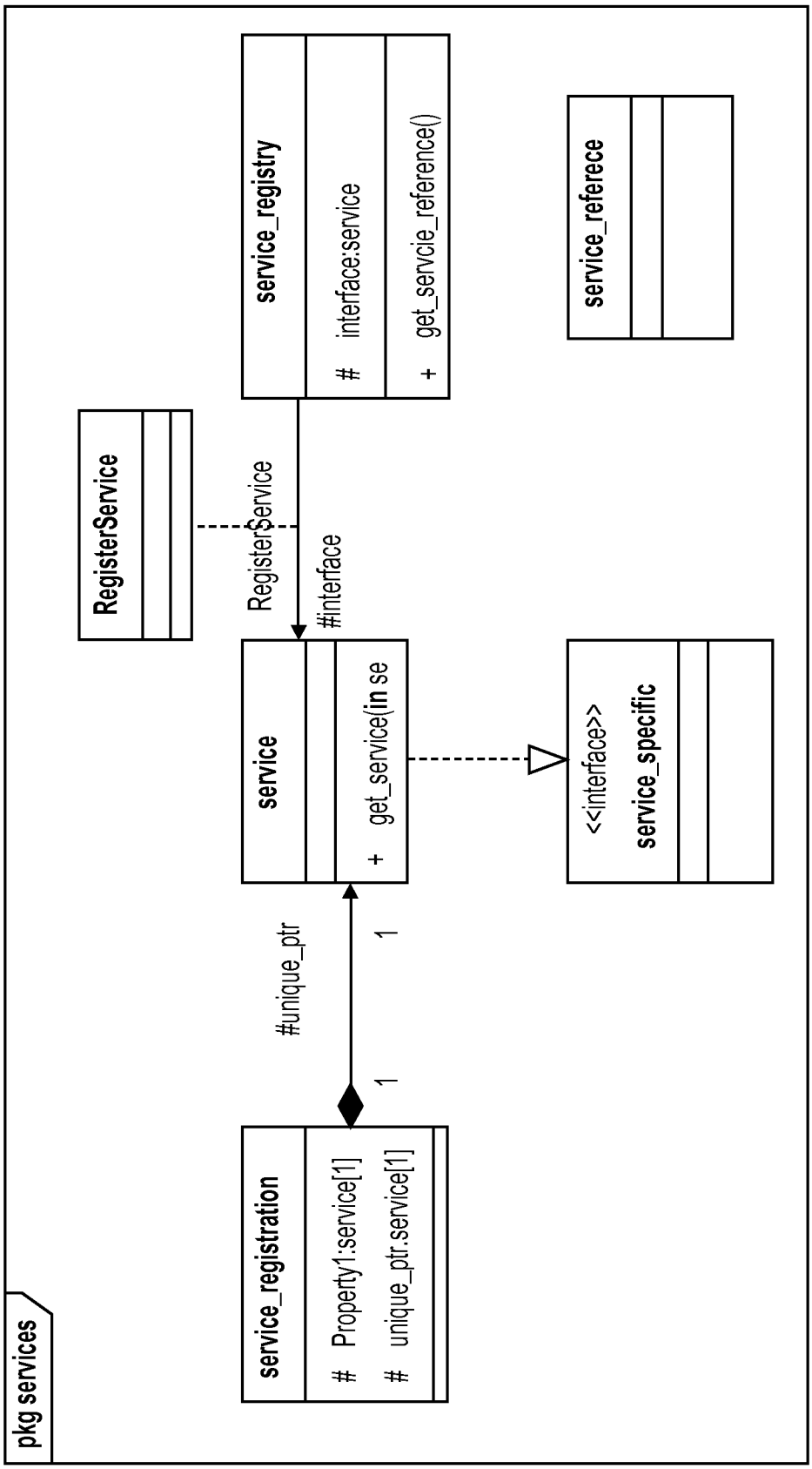

pkg services
service_registry
interface:service
+ get_servcie_reference()
service_referece
RegisterService
RegisterService
#interface
service
+ get_service(in se
<<interface>>
service_specific
#unique_ptr
1
1
service_registration
Property1:service[1]
unique_ptr.service[1]

FIG. 7

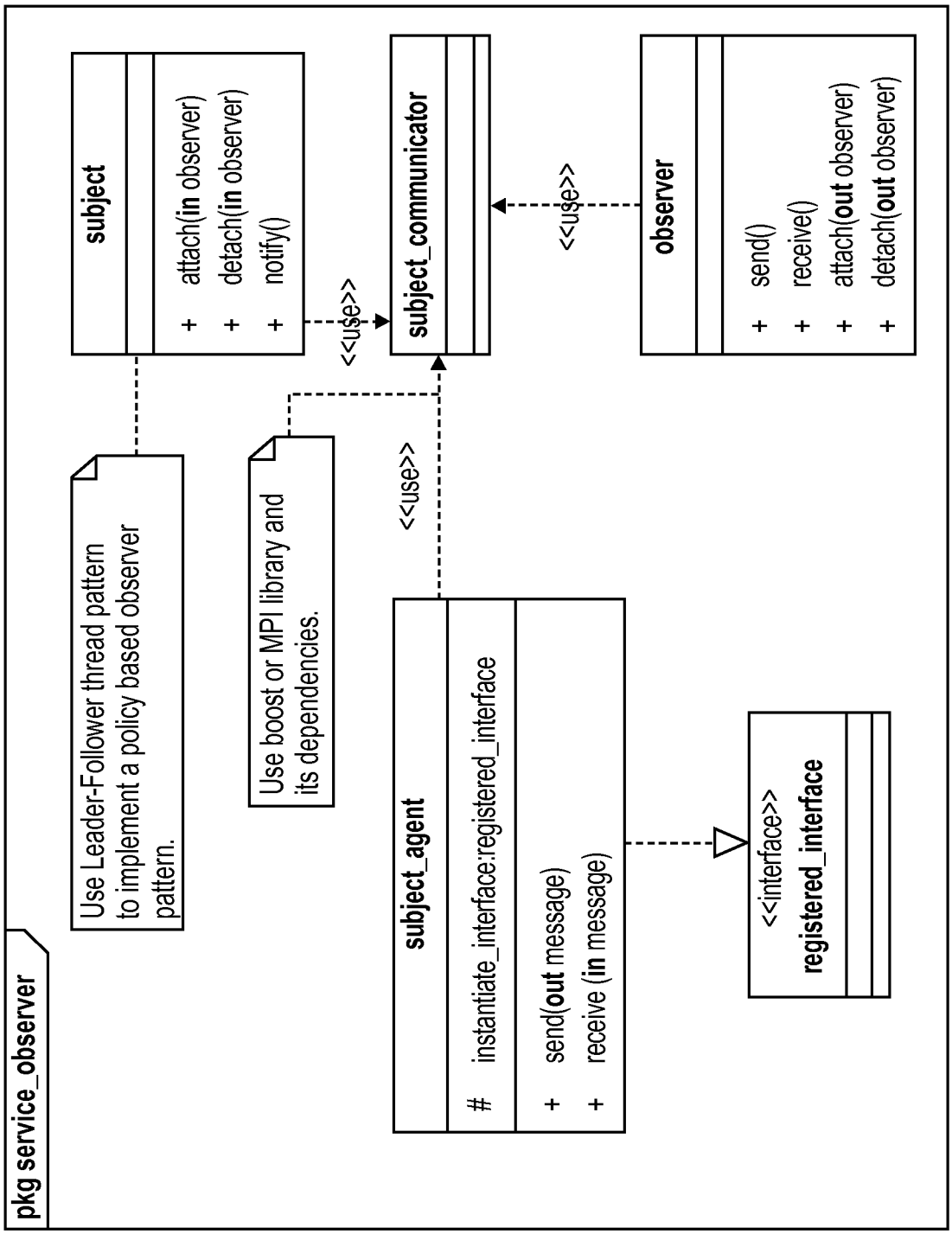

pkg service_observer
subject
+ attach(in observer)
+ detach(in observer)
+ notify()
<<use>>
subject_communicator
<<use>>
observer
+ send()
+ receive()
+ attach(out observer)
+ detach(out observer)
Use Leader-Follower thread pattern to implement a policy based observer pattern.
Use boost or MPI library and its dependencies.
<<use>>
subject_agent
instantiate_interface:registered_interface
+ send(out message)
+ receive (in message)
<<interface>>
registered_interface

FIG. 8

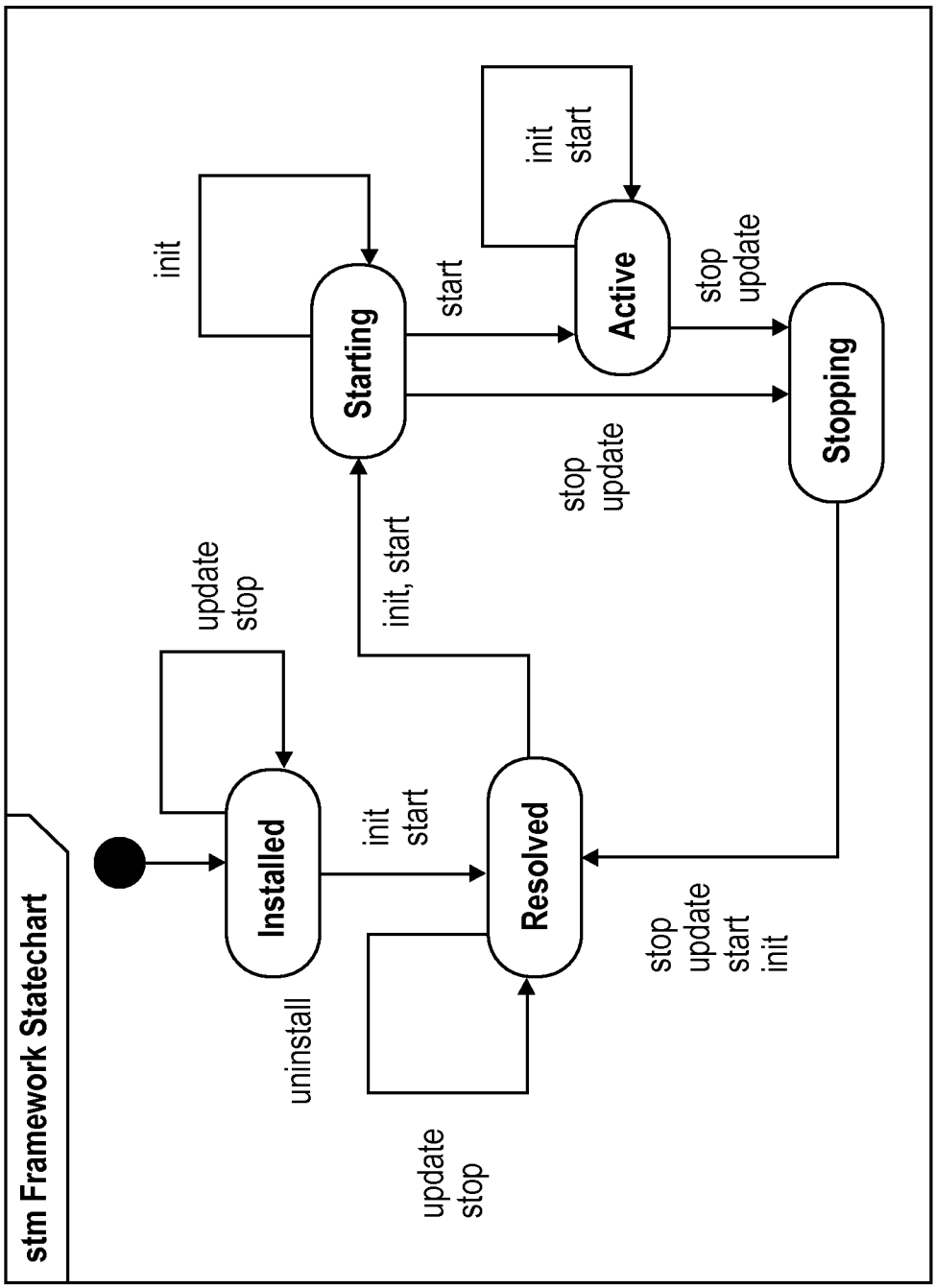

stm Framework Statechart
Installed
Resolved
Starting
Active
Stopping
init
start
stop
update
init, start
uninstall
update
stop
stop
update
start
init

FIG. 9

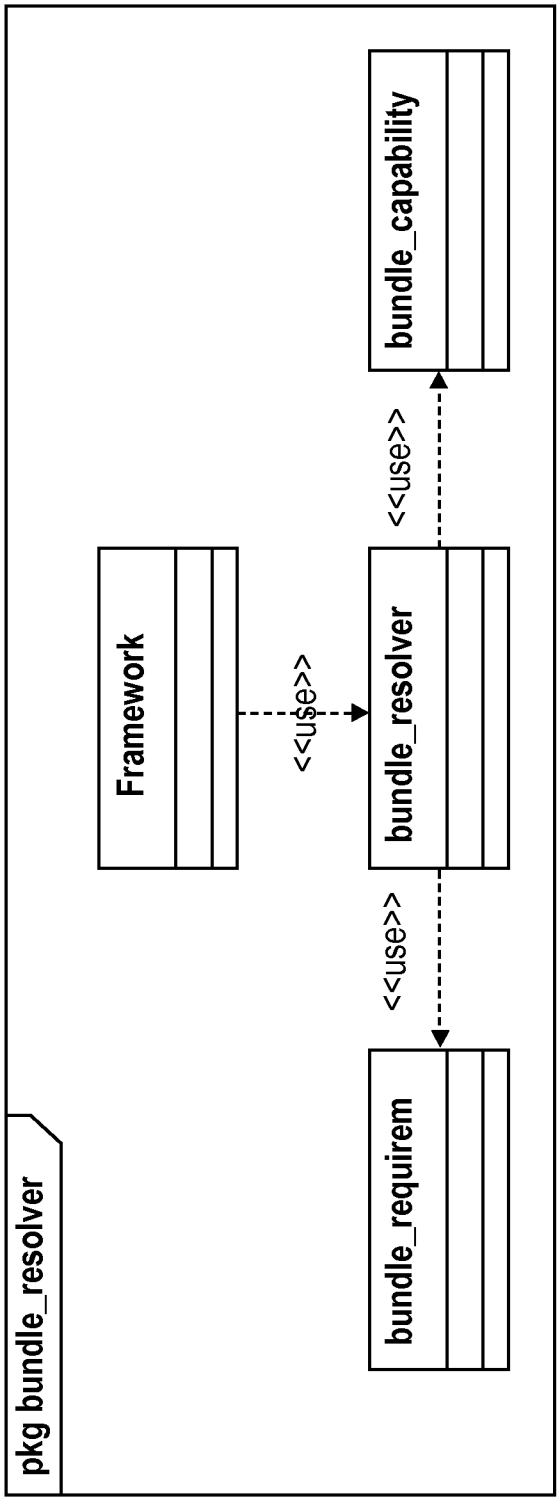

pkg bundle_resolver
Framework
bundle_resolver
bundle_capability
bundle_requirem
<<use>>
<<use>>
<<use>>

FIG. 10

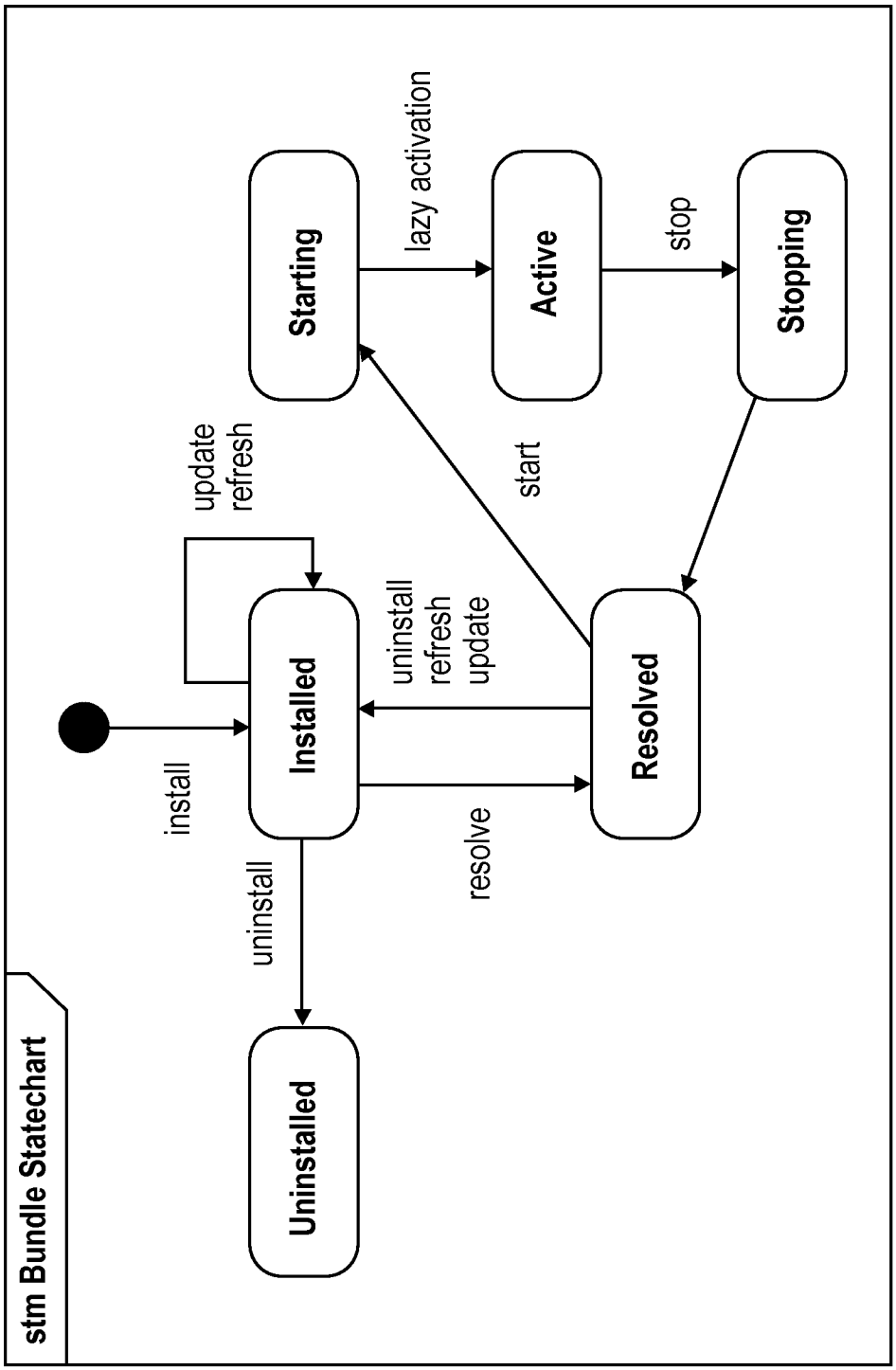

stm Bundle Statechart
Uninstalled
Installed
Starting
Active
Stopping
Resolved
install
uninstall
update
refresh
lazy activation
stop
start
uninstall
refresh
update
resolve

FIG. 11

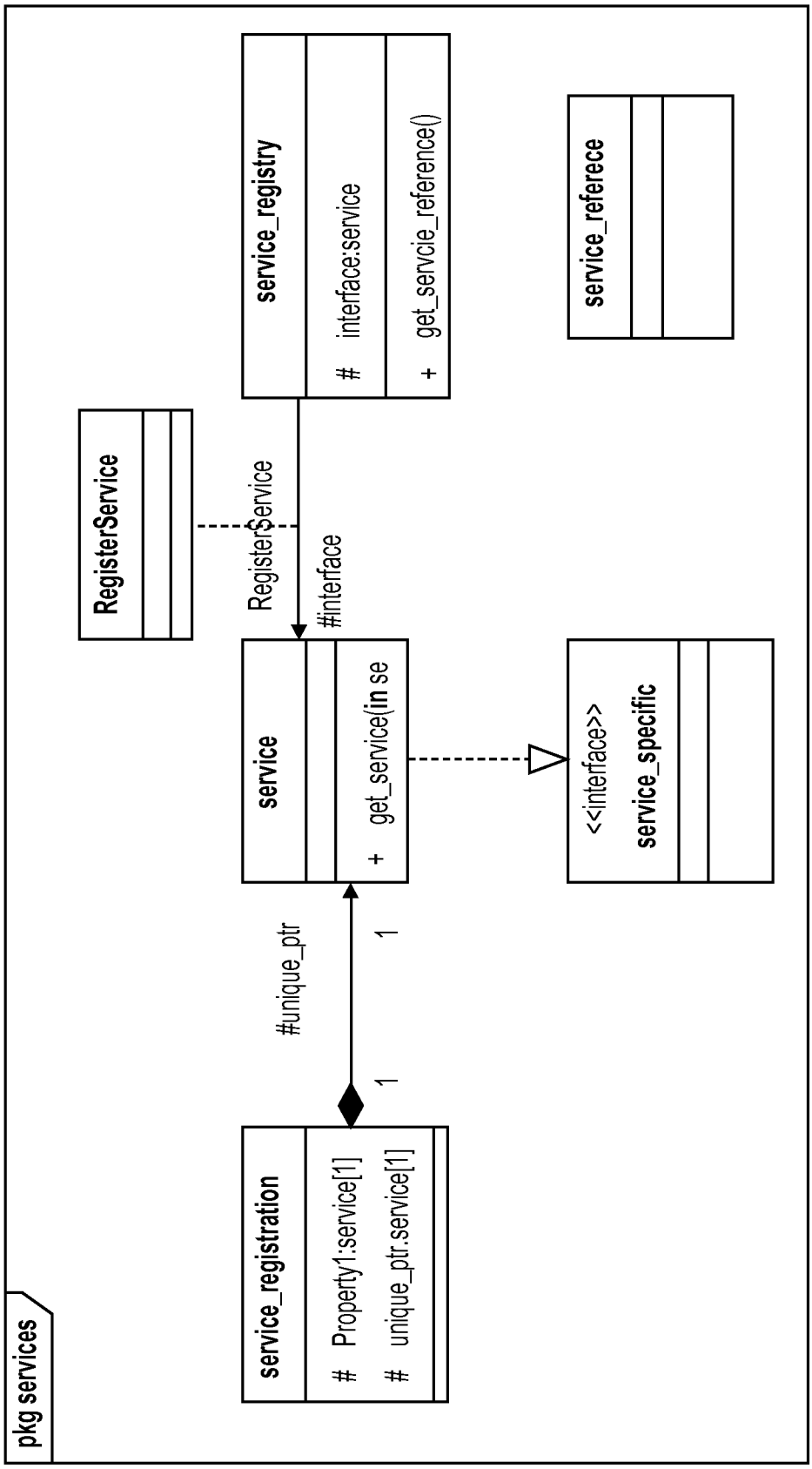

pkg services
service_registry
interface:service
+ get_servcie_reference()
service_referece
RegisterService
RegisterService
#interface
service
+ get_service(in se
<<interface>>
service_specific
#unique_ptr
1
1
service_registration
Property1:service[1]
unique_ptr.service[1]

FIG. 12

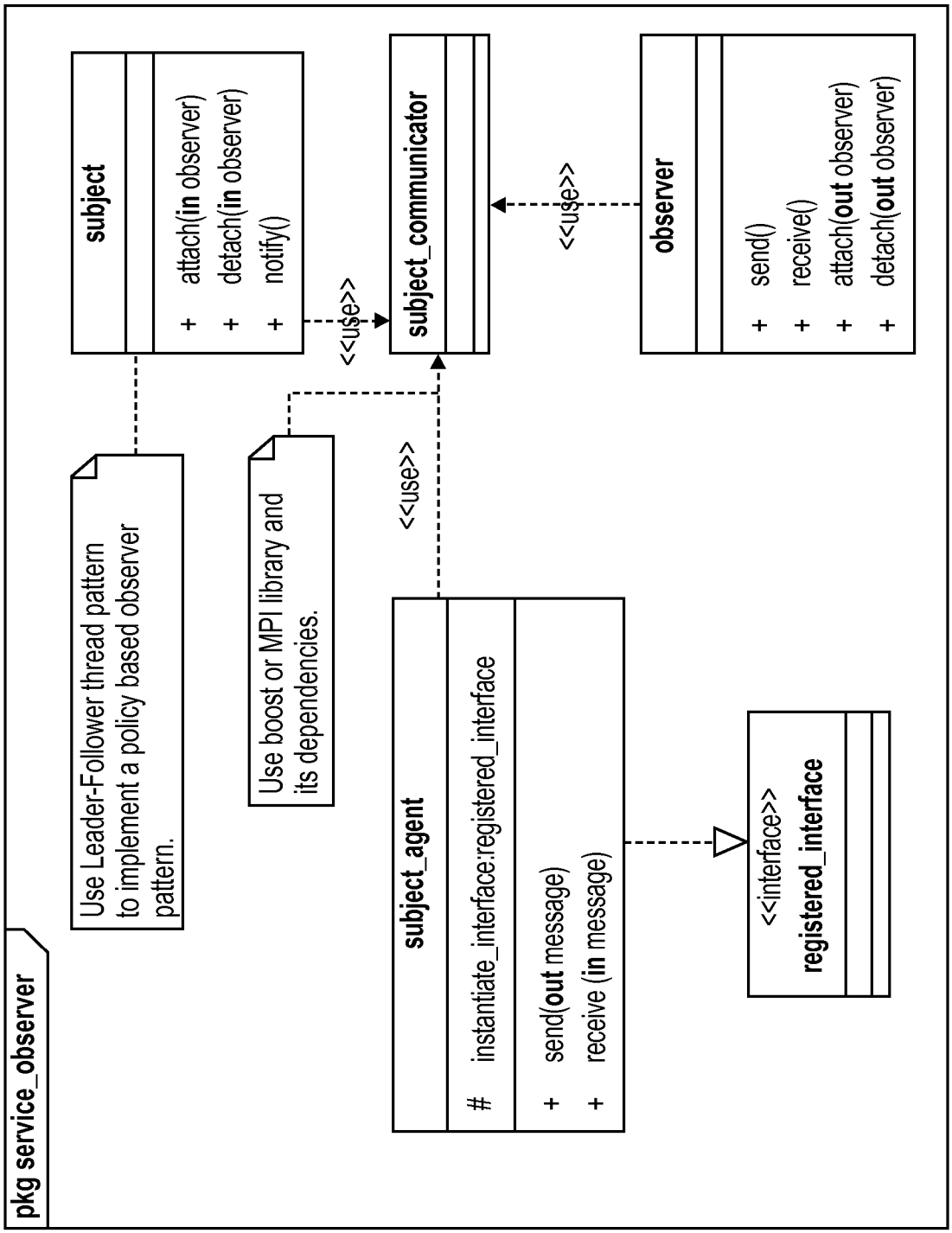

pkg service_observer
subject
+ attach(in observer)
+ detach(in observer)
+ notify()
subject_communicator
observer
+ send()
+ receive()
+ attach(out observer)
+ detach(out observer)
<<use>>
<<use>>
Use Leader-Follower thread pattern to implement a policy based observer pattern.
Use boost or MPI library and its dependencies.
<<use>>
subject_agent
instantiate_interface:registered_interface
+ send(out message)
+ receive (in message)
<<interface>>
registered_interface

FIG. 13

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2014/029839

A. CLASSIFICATION OF SUBJECT MATTER

G06F 9/30(2006.01)i, G06F 9/06(2006.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F 9/30, H04W 12/06

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
Korean utility models and applications for utility models
Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
eKOMPASS(KIPO internal) & Keywords: embedded, framework, plugin, linux, network, storage

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2009-0327996 A1 (SIEGEMUND FRANK et al.) 31 December 2009 See abstract; paragraphs [0021]-[0023],[0031]-[0064]; figures 1,4-6.	1-11
Y		12-19
Y	US 7519976 B2 (BLEVINS MIKE) 14 April 2009 See abstract; Col.3 line 12-Col.4 line 17,Col.5 line9-21,Col.7 line 16-51, Col.10 line 41 - Col.11 line 2, Col.12 line 37 - Col.15 line 42; figures 1,3,6.	12-19
A		1-11
A	US 6836796 B2 (SCHWARTZ ELLIOT et al.) 28 December 2004 See abstract; Col.9 line 6 - Col.10 line 56.	1-19
A	US 2010-0290424 A1 (COLLINGRIGE ROBERT) 18 November 2010 See paragraphs [0074]-[0076]; figure 5.	1-19
PX	US 2013-0183935 A1 (VLADIMIR HOLOSTOV et al.) 18 July 2013 See abstract; paragraphs [0041]-[0045],[0054]-[0063]; figures 3,5.	12-19
PA		1-11

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents,such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
30 July 2014 (30.07.2014)	**31 July 2014 (31.07.2014)**

Name and mailing address of the ISA/KR
International Application Division
Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon Metropolitan City, 302-701, Republic of Korea
Facsimile No. +82-42-472-7140

Authorized officer
KYUNG, Youn Jeong
Telephone No. +82-42-481-3452

Form PCT/ISA/210 (second sheet) (July 2009)

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.
PCT/US2014/029839

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2009-0327996 A1	31/12/2009	US 8661407 B2	25/02/2014
US 7519976 B2	14/04/2009	AU 2003-223760 A1	19/12/2003
		AU 2003-223760 A8	19/12/2003
		US 2004-0049481 A1	11/03/2004
		US 2004-0221261 A1	04/11/2004
		US 7424717 B2	09/09/2008
		WO 2003-102722 A2	11/12/2003
		WO 2003-102722 A3	01/04/2004
US 6836796 B2	28/12/2004	EP 1386246 A1	04/02/2004
		EP 1386246 A4	20/05/2009
		US 2002-0133581 A1	19/09/2002
		WO 02-075576 A1	26/09/2002
US 2010-0290424 A1	18/11/2010	EP 2241138 A1	20/10/2010
		US 8687547 B2	01/04/2014
		WO 2009-098432 A1	13/08/2009
US 2013-0183935 A1	18/07/2013	WO 2013-109452 A1	25/07/2013