July 2021

MECHANISM FOR CROSS PROCESS COMMUNICATION WITH ANR CHECKING AND AUTOMATIC OBJECT SERIALIZATION

Rafael Lima

Roberto Perez

Jorge Pereira

David Notario

MECHANISM FOR CROSS PROCESS COMMUNICATION WITH ANR CHECKING AND AUTOMATIC OBJECT SERIALIZATION

ABSTRACT

A host and a client may perform inter-process communication (IPC) with automatic object serialization and deserialization and with application not responding (ANR) checking. A sender (e.g., a host process, host application, host device, a client process, client application, client device, etc.) may automatically serialize an object by using reflection (e.g., the ability of a process to examine, introspect, and modify its own structure and behavior) to recursively obtain all fields and corresponding key-value pairs from each layer of the object (as well as any parent object with inheritance). The sender may bundle the fields and key-value pairs and transmit the bundle to a receiver (e.g., the client if the host is the sender, the host if the client is the sender, etc.) via a network. The receiver may examine the bundle and determine the type of the object (e.g., arrays, string classes, interfaces, etc.) based on the fields and/or the key-value pairs stored in the bundle. The receiver may then implement the appropriate process for reconstructing the object from the based on the fields and the corresponding key-pairs. This serialization and deserialization process may be performed for each communication transmitted between the sender and the receiver. Additionally or alternatively, when the sender communicates with the receiver, the sender may send a message via a binder that causes the receiver to send a generic callback to the sender. In some examples, if the sender does not receive the callback before a predetermined period expires (e.g., the call timed-out), the sender may output a notification that the receiver (e.g., the application executing at the receiver) is not responding. In this way, the binder may provide ANR checking that informs the user whether an error has occurred or not.

DESCRIPTION

This disclosure relates to inter-process communication (IPC) and/or cross-process communication in an environment in which a mechanism allows processes to communicate and synchronize actions. FIG. 1 below is a conceptual diagram illustrating a computing device 100, which may be any mobile or non-mobile computing device, such as a cellular phone (including a so-called smartphone), a desktop computer, a laptop computer, a tablet computer, a portable gaming , a portable media player, an e-book reader, a watch (including a so-called smartwatch), a gaming controller, a vehicle infotainment system, a vehicle head unit, etc. As shown in FIG. 1, computing device 100 may include one or more processors 104, and one or more storage devices 106. Storage devices 106 may include a client bundling module 102, an operating system 108 ("OS 108"), one or more application(s) 110, and a data repository 112.

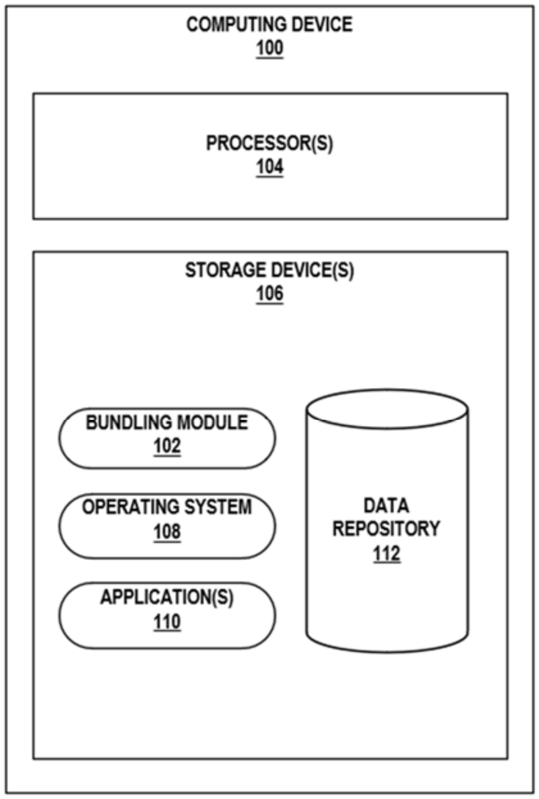

FIG. 1

Processors 104 may implement functionality and/or execute instructions associated with computing device 100. Examples of processors 104 may include one or more of an application specific integrated circuit (ASIC), a field programmable gate array (FPGA), an application processor, a display controller, an auxiliary processor, a central processing unit (CPU), a graphics processing unit (GPU), one or more sensor hubs, and any other hardware configure to function as a processor, a processing unit, or a processing . Processors 104 may retrieve and execute instructions stored by storage devices 106 that cause processors 104 to perform the operations described in this disclosure.

Storage devices 106 may include one or more computer-readable storage media. For example, storage devices 106 may be configured for long-term, as well as short-term storage of information, such as instructions, data, or other information used by computing device 100. In some examples, storage devices 106 may include non-volatile storage elements. Examples of such non-volatile storage elements include magnetic hard discs, optical discs, solid state discs, and/or the like. Examples of volatile memory s include random-access memories (RAM), dynamic random-access memories (DRAM), static random-access memories (SRAM), etc.

A developer may design software to be backwards compatible to allow for interoperability with an older legacy system. For example, the developer may define a protocol for serializing (e.g., a process of translating an object into a format that can be stored or transmitted) and deserializing (e.g., a process of using data structured from a format to rebuild an object) messages being communicated between a host (e.g., a host process, host application, host device, etc.) and a client (e.g., client process, client application, client device, etc.) to implement backwards compatibility. For example, a developer may define the serialization protocol such that, when the host and client pass messages containing objects, the host and client are aware of the existence of all the fields of each object, thereby ensuring the message is correctly read by both sides. Defining the protocol for serialization and deserialization can avoid problematic situations where a communicated message contains fields that a receiver (e.g., the host if the client is the sender or the client if the host is the sender) is not aware exists. However, manually defining the protocol for serialization and deserialization may be complex and time-consuming, potentially increasing the cost of software development.

In accordance with techniques of this disclosure, computing device 100 may perform IPC with automatic object serialization and deserialization. In some examples, bundling module 102 of computing device 100 may enable serialization of an object (to be communicated between a host and a client) stored in data repository 112 by using reflection (e.g., the ability of a process to examine, introspect, and modify its own structure and behavior) to recursively obtain all fields and corresponding key-value pairs from each layer of the object (as well as any parent object with inheritance). Bundling module 102 may then bundle all the fields and corresponding key-value pairs together in a message for a sender (e.g., the host) to send to a receiver (e.g., the client). Responsive to receiving the message containing the bundle, the receiver may examine (e.g., via bundling module 102) the bundle to identify all the fields and corresponding key-value pairs and reconstruct the object (e.g., according to a deserialization protocol), in this way addressing backwards compatibility issues.

In some examples, during a first iteration of the recursion, bundling module 102 may obtain each field and corresponding key-value pair of the outermost layer of the object. During subsequent iterations, bundling module 102 may apply the same process to the inner layers of the object, deconstructing the object from the outermost layer to the innermost layer (e.g., layer-by-layer) until the values of the key-value pairs are primitive values (e.g., parcelable, serializable, etc.). In some examples, the primitive values may be strings, numbers, Booleans, and/or the like. Responsive to obtaining all the fields and key-value pairs of the object, bundling module 102 may store all the fields and key-value pairs into a bundle.

A sender (e.g., a host, a client, etc.) may transmit the bundle of fields and key-value pairs to a receiver (e.g., the client if the host is the sender, the host if the client is the sender, etc.). Bundling module 102 may examine the bundle and determine the type of the object (e.g., arrays, string classes, interfaces, etc.) based on the fields stored in the bundle and/or the key-value pairs. Responsive to determining the type of the object, bundling module 102 may implement the appropriate process for reconstructing the object from the innermost layer to the outermost layer (e.g., layer-by-layer) based on the fields and the corresponding key-pairs. For example, bundling module 102 may populate the fields of the innermost layer of the object with the primitive values of the key-value pairs and then recursively proceed in this manner until the entire object is rebuilt. In some examples, the sender and receiver may perform this serialization and deserialization process for each communication.

As such, bundling module 102 may facilitate backwards compatibility. For example, the sender in the above example may be OS 108 (e.g., a host process) that is the most current version of the OS, and the receiver may be application 110 (e.g., a client process) that was written for an older version of the OS. While this may normally result in issues due to messages containing fields that application 110 is not aware exists, the automatic serialization and deserialization performed by bundling module 102 ensures that communicated objects are deconstructed and reconstructed (e.g., from the innermost layer to the outermost layer) based on the fields and the corresponding key-pairs in the bundle created by bundling module 102.

In another example, the sender in the above example may be a library that is the most current version of the library, and the receiver may be application 110 that incorporates an older version of the library. Again, the automatic serialization and deserialization performed by bundling module 102 ensures that communicated objects are deconstructed and reconstructed based on the fields and the corresponding key-pairs in the bundle created by bundling module 102, avoiding problems that otherwise may cause crashes, bugs, etc.

In general, IPC may be synchronous (e.g., a mechanism that ensures multiple processes join up or handshake at a certain point) or asynchronous. For example, with synchronous IPC, if a receiver (e.g., application 110) is slow responding to a message from a sender (e.g., OS 108), the sender may block a process until the receiver responds. Conversely, with asynchronous IPC, the sender may continue the process without waiting for the receiver to receive (and respond to) the message. Because synchronous IPC may cause user interface components and/or other components to appear unresponsive to a user (e.g., a receiver may not respond due to an application not responding (ANR), causing the sender to block a process until the thread is killed), a developer may design a process to use asynchronous IPC. However, in some cases, a mechanism for coordinating concurrent activity is desirable, such that a conventional implementation of asynchronous IPC is not appropriate.

In accordance with techniques of this disclosure, a sender may communicate with a receiver via a binder such that whenever the sender sends a message to the receiver, the receiver sends a generic callback to the sender. Thus, if synchronous behavior is expected, the sender may block one or more processes until the receiver responds with a generic callback (e.g., within a predetermined period). For example, if OS 108 sends a message to application 110 via the binder, application 110 may send a generic callback to OS 108. If OS 108 does not receive the generic callback before a predetermined period expires, OS 108 may determine application 110 is not responding. In this way, the binder may provide ANR checking that informs a user of computing device 100 whether an error has occurred or not.

In some examples, the generic callback may contain the data requested by the sender or an acknowledgement of processing to confirm that the call by the sender was dispatched to the correct address space. Accordingly, even if asynchronous behavior is expected, the sender may still communicate with the receiver via the binder to cause the receiver to respond with the generic callback, which sender may use for various operations.

In some examples, responsive to receiving the generic callback, the sender may generate a log of the success and failure rate of calls. The log may be used to create metrics for tracking the performance of software, such as APIs (e.g., stored in storage devices 106, storage devices 106) being called by the sender. The log may also be used to identify issues, such as whether a library (e.g., stored in data repository 112 or stored elsewhere) is crashing, whether the telemetry is malfunctioning, whether the generic callback is insufficient (such that a specific callback is required), etc.

One or more advantages of the techniques described in this disclosure include providing a method for automatic object serialization and deserialization. Automation of object serialization and deserialization may decrease the cost of software development by saving developers time, especially with respect to implementing backwards compatibility of software. In addition, the techniques described in this disclosure provide a mechanism (e.g., the binder) for coordinating concurrent activity (e.g., synchronous IPC), even if a process uses asynchronous IPC by default. In some examples, the binder may enable ANR checking by causing a receiver of a communication to automatically send a generic callback within a predetermined period (where non-receipt of the generic callback with in a predetermined period indicates that the receiver is not responding). The generic callbacks may contain information indicative of the success and failure rate of calls (e.g., of APIs), which may be used to track performance of software and troubleshoot software and/or hardware issues.

Although described here with respect to a host process and a client process, it should be understood that techniques of this disclosure are not so limited. That is, the techniques may be applied to a host application and a client application, a host device and a client device, and so on. Thus, in some examples, the techniques may be applied to a distributed environment in which the host and the client do not share the same address space.

It is noted that the techniques of this disclosure may be combined with any other suitable technique or combination of techniques. As one example, the techniques of this disclosure may be combined with the techniques described in U.S. Patent Application Publication No. 2019/0370091A1. In another example, the techniques of this disclosure may be combined with the techniques described in U.S. Patent Application Publication No. 2006/0075304A1. In yet another example, the techniques of this disclosure may be combined with the techniques described in U.S. Patent Application Publication No. 2020/0184140A1.