dbus: a brief introduction

Bonnie King
Fermi Linux Users Group Meeting
September 26, 2012

dbus

An inter-process communication system (IPC) used to pass messages:

dbus

An inter-process communication system (IPC) used to pass messages:

- between desktop applications within the same session (*session daemon*)

dbus

An inter-process communication system (IPC) used to pass messages:

- between desktop applications within the same session (*session daemon*)
- between desktop applications and the operating system (*system daemon)*

dbus

An inter-process communication system (IPC) used to pass discrete messages:

- between desktop applications within the same session (*session daemon*)
- between desktop applications and the operating system (*system daemon)*
- between any two applications, bypassing the message bus daemon (*in theory...*)

dbus

Designed as a middleware layer for KDE, Gnome... used by:

- udev, HAL
- NetworkManager
- GNOME Screen Saver, Power Manager...
- Lots of things that start with "K"
- Avahi
- ...

communication

Communication is via unencrypted UNIX sockets between apps and daemons on the same host.

Stateful and connection-based like TCP, but not for data streams.

dbus-daemon

The message bus executable built on **`libdbus`** which routes messages to applications using a publish/subscribe paradigm.

libdbus

The low-level C API:

"If you use this low-level API directly, you're signing up for some pain."

http://dbus.freedesktop.org/doc/api/html/index.html

wrapper libraries

- Glib
- Python
- Qt
- Java
- ...

object paths/namespaces

libdbus provides object paths so higher level bindings can name their object instances.

These objects are exported for use by other applications.

```
'/org/freedesktop/NetworkManager/Devices/eth0'
'/org/freedesktop/Hal/Manager'
'/com/example/WordProcessor'
```

object members

- methods:
  - can be invoked on an object, perhaps with parameters, may return output
- signals
  - "broadcasts"; may include data. Publish events to anyone who cares.

interfaces

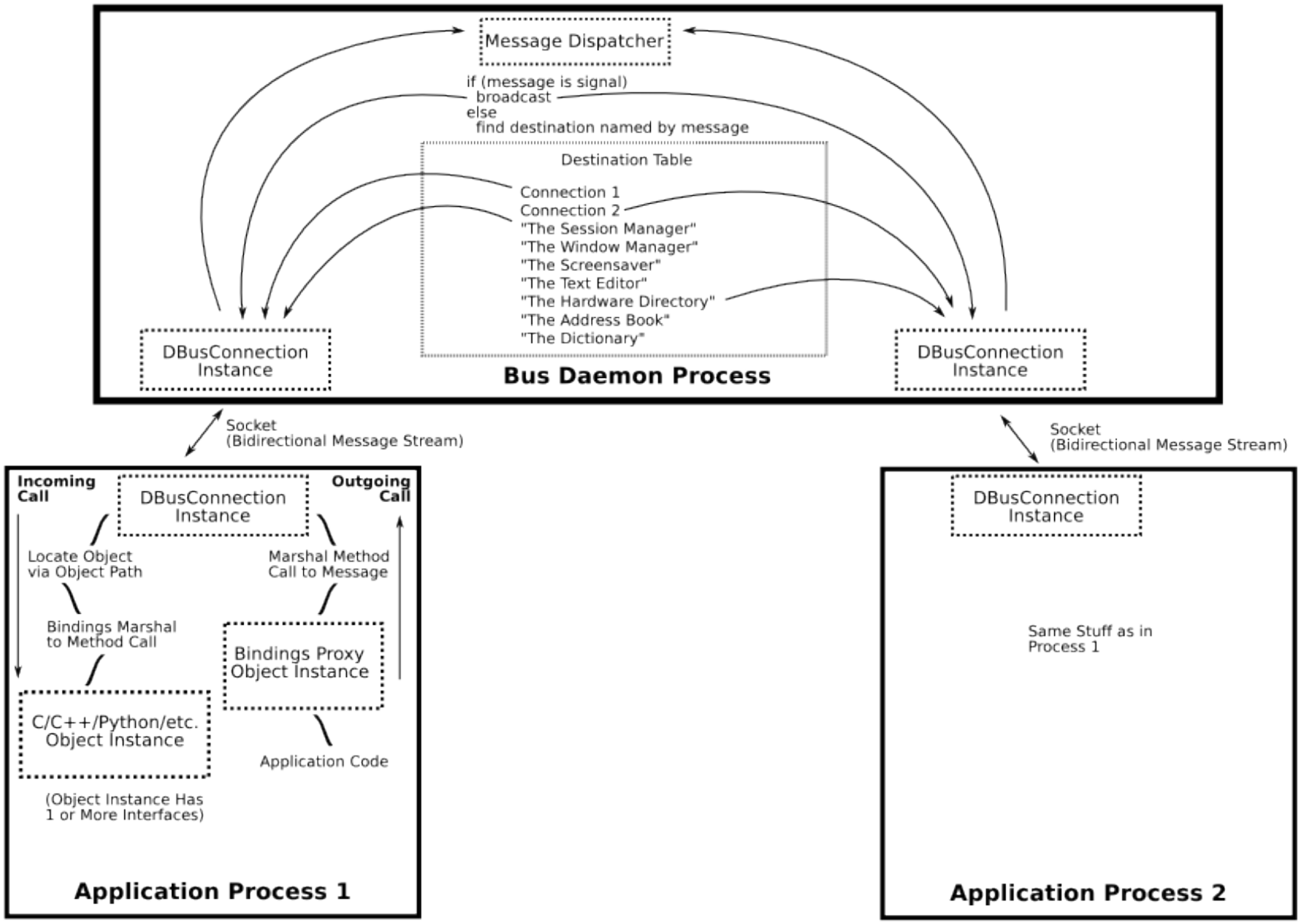
Groups of methods and/or signals provided by a dbus object.

proxy objects

Your high-level API interacts with the remote dbus objects using a proxy object.

You interact with this native object as if it were the "real" object on the across the message bus, but don't have to worry about message processing.

from http://dbus.freedesktop.org/doc/dbus-tutorial.html



example: the helloservice

http://excid3.com/blog/an-actually-decent-python-dbus-tutorial/

```
#!/usr/bin/python

import gtk
import dbus
import dbus.service
from dbus.mainloop.glib import DBusGMainLoop

class MyDBUSService(dbus.service.Object):
      def __init__(self):
          bus_name = dbus.service.BusName('org.foo.helloservice', bus=dbus.SessionBus())
          dbus.service.Object.__init__(self, bus_name, '/org/foo/helloservice')

      @dbus.service.method  org.foo.helloservice
      def hello(self):
          return  Hello, World!

DBusGMainLoop(set_as_default=True)
myservice = MyDBUSService()
gtk.main()
```

the .service file

In `/usr/share/dbus-1/services:`

```
# cat org.foo.helloservice.service:
[D-BUS Service]
Name=org.foo.helloservice
Exec=/home/bonniek/bin/hello_service.py
```

No need to restart the service after adding a .service file.

the helloclient

```
#/usr/bin/python


import dbus

bus = dbus.SessionBus()
helloservice = bus.get_object('org.foo.helloservice', '/org/foo/helloservice')
hello = helloservice.get_dbus_method('hello', 'org.foo.helloservice')
print hello()
```

or with dbus-send...

```
# dbus-send --print-reply --dest=org.foo.helloservice /org/foo/helloservice org.
foo.helloservice.hello
method return sender=:1.351 -> dest=:1.363 reply_serial=2
   string "Hello,World!"
```

example: register to receive signals

```
#!/usr/bin/python

import gtk
import dbus
import dbus.service
from dbus.mainloop.glib import DBusGMainLoop

def newdev_message(message):
    print  "Plug and Play! %s" %message"

bus = dbus.SystemBus()
bus.add_signal_receiver(newdev_message, dbus_interface='org.freedesktop.Hal.Manager',
                        signal_name='DeviceAdded')


DBusGMainLoop(set_as_default=True)
gtk.main()
```

introspection

Return an XML description of an interface:

```
# dbus-send --system --print-reply --dest=org.freedesktop.
Hal /org/freedesktop/Hal/Manager org.freedesktop.DBus.
Introspectable.Introspect
```

qdbus

List servicenames that are running:

```
# qdbus
# qdbus --system
```

Get list of methods on a dbus interface:

```
# qdbus org.gnome.ScreenSaver /Screensaver
```

dbus-monitor

Monitor a message bus:

```
Usage: dbus-monitor [--system | --session | --
address ADDRESS] [--monitor | --profile ] [watch
expressions]

# dbus-monitor
# dbus-monitor --system
```

resources

- http://dbus.freedesktop.org
- http://code.google.com/p/dbus-tools/wiki/DBusCli (more user-friendly than dbus-send)
- http://www.freedesktop.org/wiki/Software/DbusProjects