Kiwi PyCon 2017 – Auckland New Zealand

Client / Server Programming Model
Desktop-Bus Communication
pydbus python library
...and systemd

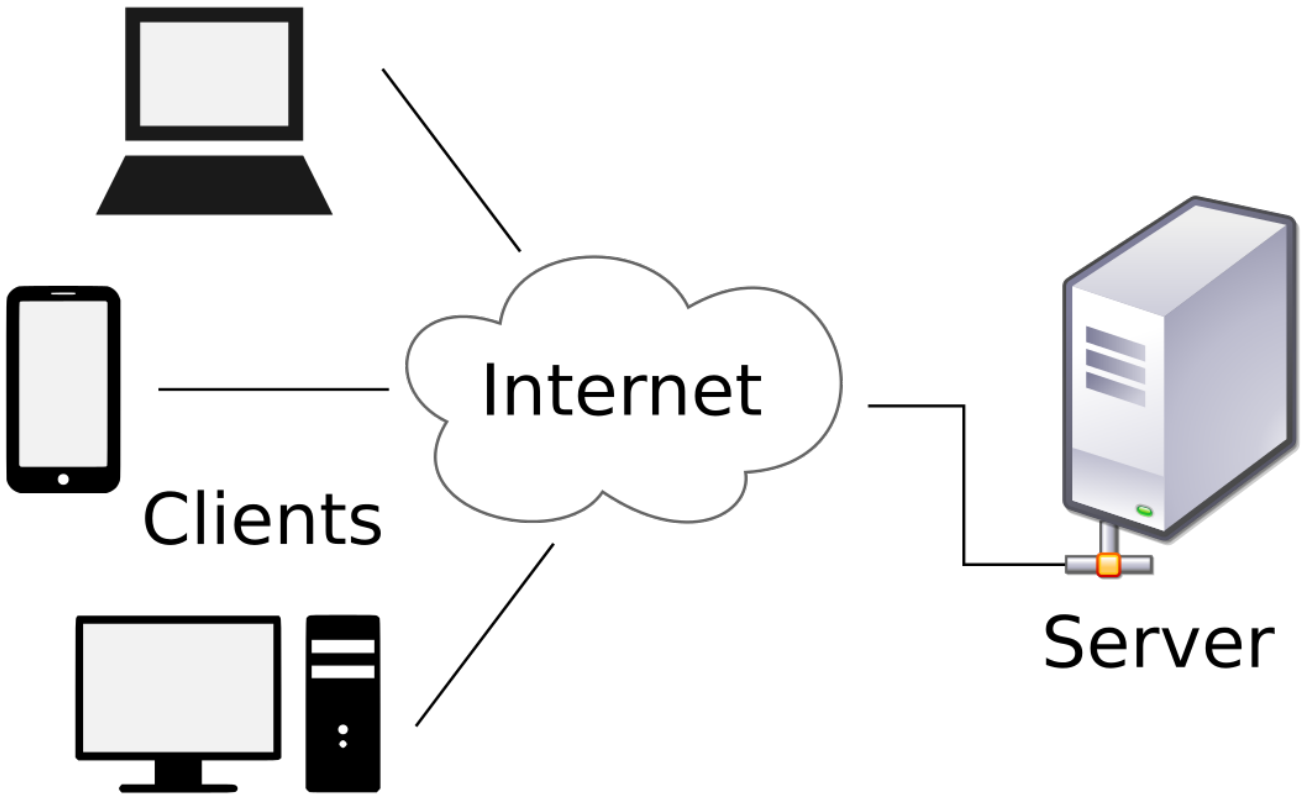
Presenter: Ian Stewart – Hamilton Python User Group (HamPUG)
Email: stwrtn@gmail.com
Website: https://hampug.pythonanywhere.com/
Posted at: https://github.com/irsbugs/kiwipycon2017

This presentation is designed to also be a reference document.

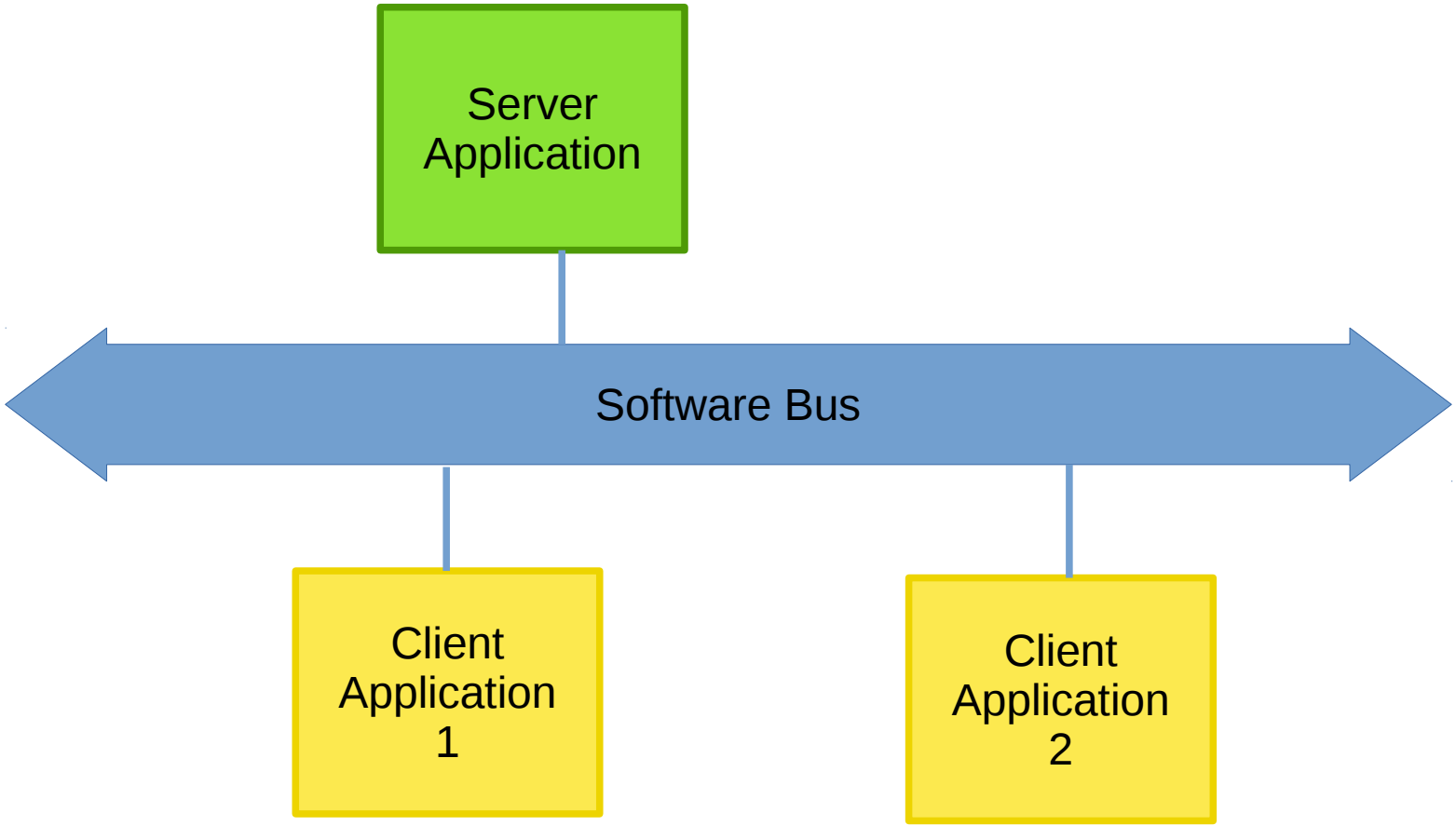
Contents of Presentation

- Introduce: Client/Server, D-Bus, pydbus, and gdbus tool
- Developing Session D-Bus Server applications for Method Calls and testing with gdbus tool
- Developing Session D-Bus Client Applications for Method Calls
- Server Emitting a Session D-Bus Signal
- Client subscribing to a Session D-Bus Emitted Signal
- Demo
- Introduce: The System D-Bus
- Setup and demo the System D-Bus
- Setup systemd
- Client / Server Demo using GUI clients
- Practical Example
- Questions

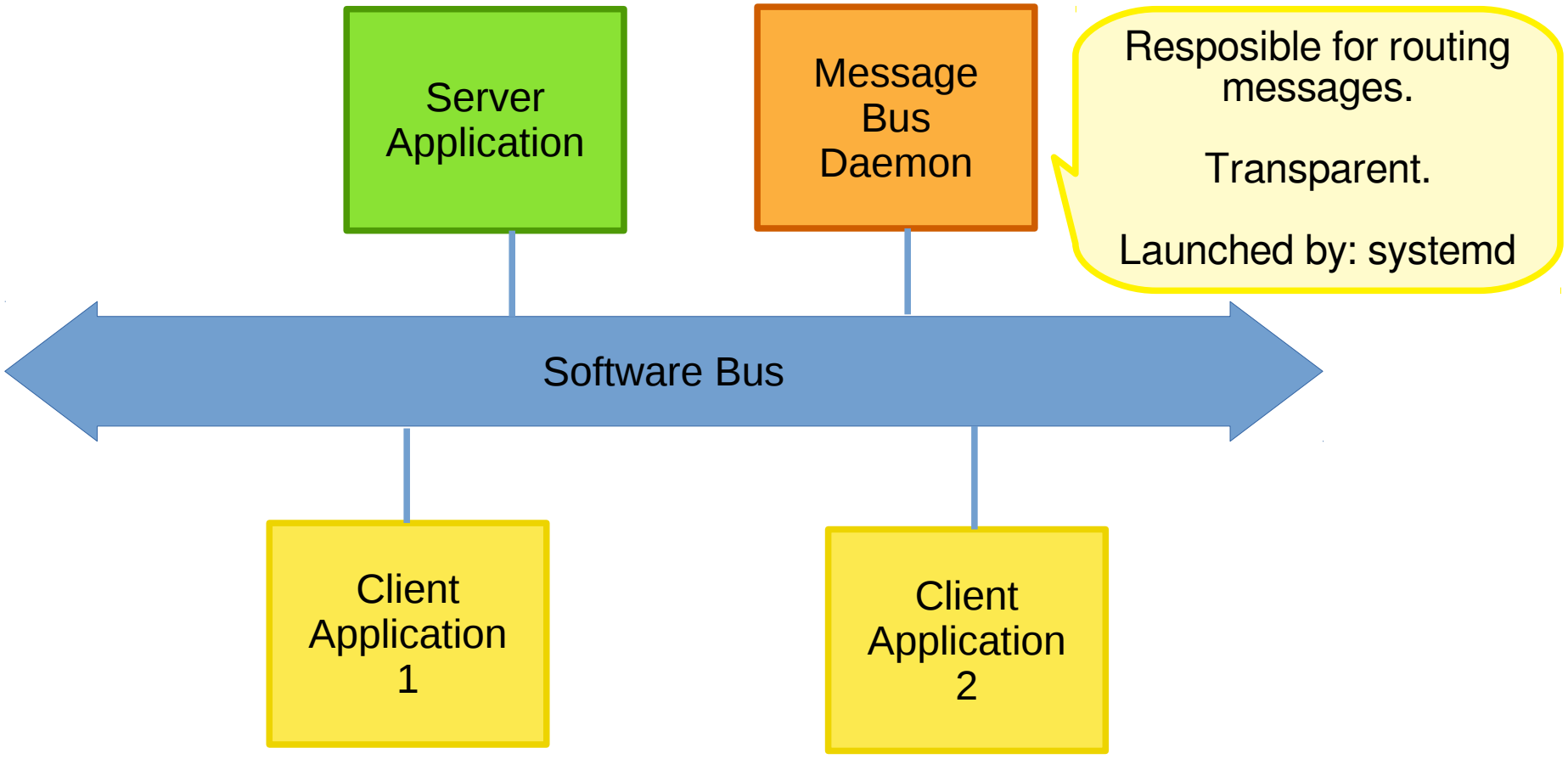
- Client / Server Model developed in 1960/70's as part of ARPANET.



Client / Server applications. Internal Software Bus



Client / Server applications. Internal Software Bus



Desktop Bus - D-Bus

- A software bus that allows computer programs concurrently running on the same machine to communicate.
- First stable release: November 2006. 11 years ago. Rev 0.12
- Initiated by Havoc Pennington and Alex Larsson from Red Hat, and Anders Carlsson.
- Specification at Rev 0.31. 29 June 2017: https://dbus.freedesktop.org/doc/dbus-specification.html

Desktop Bus - D-Bus

- The freedesktop.org project developed a free and open-source software library called **libdbus**.
- Two types of D-Bus. System Bus and Session Bus.
- Implementations: GDBus, QtDBus, sd-bus, kdbus, more...
- GDBus uses the GObject based library GIO (C code).
- PyGObject is a Python package which provides bindings for GObject, thus for GIO.
- pydbus is a python wrapper accessing PyGObject python for GIO.

- Creator: Linus Lewandowski. Warsaw – Poland.
- Home page: https://github.com/LEW21/pydbus
- Wrapper based on using gdbus functions from the python GIO library.

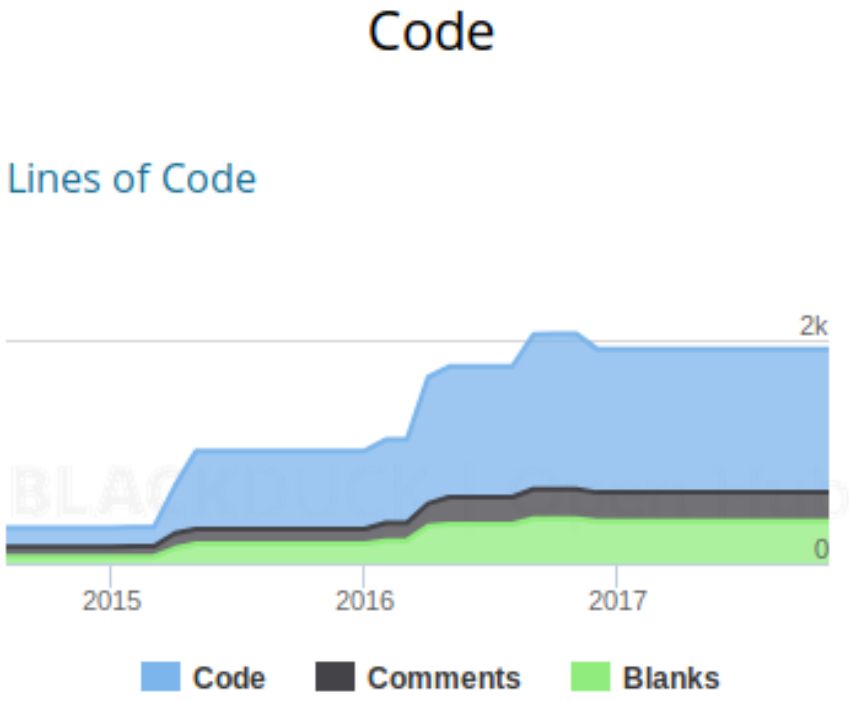
Installation of MIT licensed pydbus:

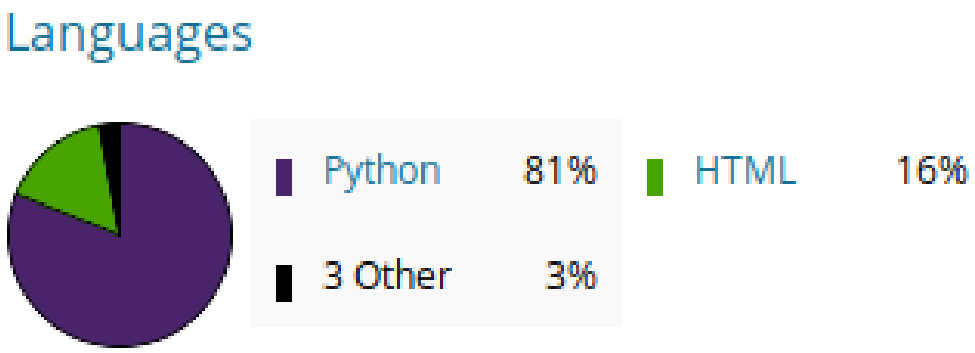
- $ sudo apt install python3-pydbus
- $ sudo pip3 install --target /usr/lib/python3/dist-packages pydbus

pydbus files located in: /usr/lib/python3/dist-packages/pydbus/
18 files @ average file size 1.8KBytes

auto_names.py	__init__.py	proxy_signal.py
bus_names.py	_inspect3.py	publication.py
bus.py	method_call_context.py	registration.py
exitable.py	proxy_method.py	request_name.py
generic.py	proxy_property.py	subscription.py
identifier.py	proxy.py	timeout.py

pydbus analysis





Analysis 20 November 2017. By BlackDuck Open Hub: https://www.openhub.net/p/pydbus

November 2017
Code: 1287
Comments: 257
Blanks: 379

pydbus – Why?

- High level.
- Simple to implement.
- Many aspects of D-Bus are automatically assigned.
- Developer can focus on the Client and Server code not the interprocess communication code.
- Modular software development.
- Data Integrity on the D-Bus seesm to be assured.

Clients and Servers on a dbus.
Layers on a server application

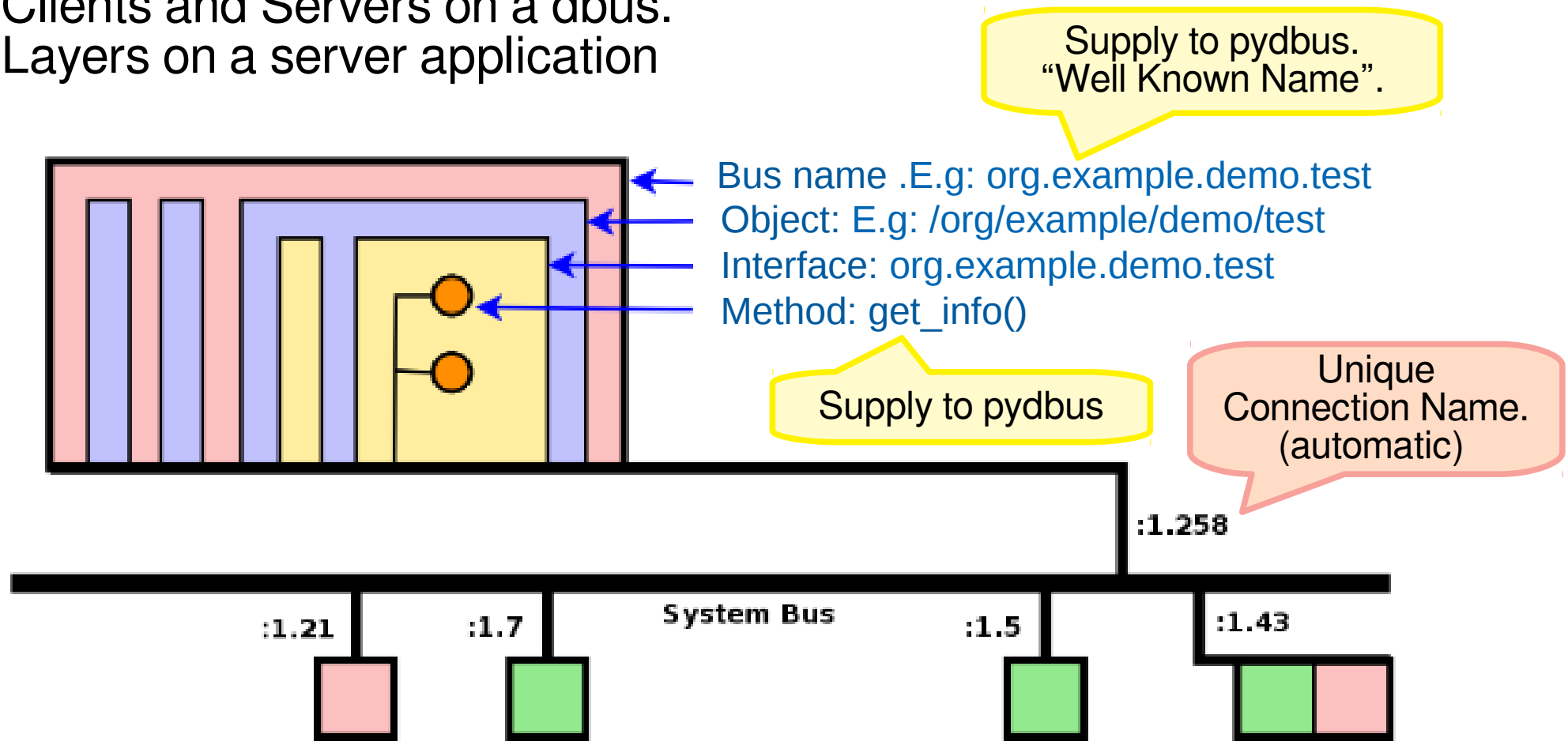


Diagram Credit: https://aleksander.es/data/GNOMEASIA2014%20-%20Introduction%20to%20DBus.pdf

gdbus tool

.

gdbus is a tool / utility included with linux distributions.

Features of gdbus tool:
- Simulate method calls to a server.
- Debug the server before invoking the client application.
- Monitor signals emitted by a server.

Test client or server code in isolation.

Reference:
https://developer.gnome.org/gio/stable/gdbus.html
https://github.com/GNOME/glib/blob/master/gio/gdbus-tool.c

Developing a Server application

.

Method calls to a server application.

Four types of method calls the client can issue to the server through the D-Bus:

- Server method has no arguments. E.g. Stop, Play, Rewind, etc.
- Server method sends data to the client.
- Server method receives data from the client.
- Server method receives data from and sends to the client.

The data type is static. For examples...
type='s' for string data
'i' for 32 bit signed integer, 'd' for double precision floating point, 'as' for a list if strings, etc.

For more static types:
https://dbus.freedesktop.org/doc/dbus-specification.html

server_demo_1.py – Beginning and End Sections.

```
# Importing...
from pydbus import SessionBus  # SystemBus
from gi.repository import GLib

# Variables / Constants / Instantiation...
bus = SessionBus()  # SystemBus
BUS = "org.example.demo.test"
loop = GLib.MainLoop()
message_count = 0
```

1. Session or System Bus

2. Instantiate

3. Well Known Name

1. Glib loop - instatiate

```
if __name__ == "__main__":
    bus.publish(BUS, DBusService_XML())
    loop.run()
```

4. publish

2. Glib loop - instatiate

server_demo_1.py – Middle Section. Method Call performs action

```
class DBusService_XML():
    """
    DBus Service XML definition.
    type="i" for integer, "s" string, "d" double, "as" list of string data.
    """
    dbus = """
    <node>
        <interface name="{}">
            <method name='server_no_args'>
            </method>
        </interface>
    </node>
    """.format(BUS)

    def server_no_args(self):
        "No arguments over the dbus. Server produces a message on the console."
        global message_count
        print("This is message {}".format(message_count))
        message_count +=1
        return
```

5. XML defines method: server_no_args

6. Class Method: server_no_args

server_demo_1.py – gdbus tool testing method call

```
$ gdbus call --session --dest org.example.demo.test --object-path /org/example/
demo/test --method org.example.demo.test.server_no_args
()
$ gdbus call --session --dest org.example.demo.test --object-p          e/
demo/test --method org.example.demo.test.server_no_args
()
$ gdbus call --session --dest org.example.demo.test --object-path /org/example/
demo/test --method org.example.demo.test.server_no_args
()
$ gdbus call --session --dest org.example.demo.test --object-path /org/example/
demo/test --method org.example.demo.test.server_no_args
()
$
```

Testing Console

Received empty tuple

```
$ python3 server_demo_1.py
This is message 0
This is message 1
This is message 2
This is message 3
```

Server Console

server_demo_2.py – Middle Section – Method Call sends data

```
class DBusService_XML():
    """
    DBus Service XML definition.
    type="i" for integer, "s" string, "d" double, "as" list of string data.
    """
    dbus = """
    <node>
        <interface name="{}">
            <method name="get_time_stamp">
                <arg type="d" name="response" direction="out">
                </arg>
            </method>
        </interface>
    </node>
    """.format(BUS)

    def get_time_stamp(self):
        "Return a Unix time. Seconds since epoch"
        return time.time() #
```

Class Method: get_time_stamp

Send out time stamp as double precision

server_demo_2.py – gdbus tool testing method call

```
$ python3 server_demo_2.py
```

Server Console

```
$ gdbus call --session --dest org.example.demo.test --object-path /org/example/
demo/test --method org.example.demo.test.get_time_stamp
(1511428204.1218235,)

$ gdbus call --session --dest org.example.demo.test --object-path /org/example/
demo/test --method org.example.demo.test.get_time_stamp
(1511428206.5049083,)

$ gdbus call --session --dest org.example.demo.test --object-path /org/example/
demo/test --method org.example.demo.test.get_time_stamp
(1511428208.3114276,)
```

Testing Console

Receive tuple with time stamp

server_demo_3.py – Middle Section – Method Call receives data

```
class DBusService_XML():
    """
    DBus Service XML definition.
    type="i" for integer, "s" string, "d" double, "as" list of string data.
    """
    dbus = """
    <node>
        <interface name="{}">
            <method name="greeting">
                <arg type="s" name="person" direction="in">
                </arg>
            </method>
        </interface>
    </node>
    """.format(BUS)

    def greeting(self, name):
        "Return Hello and the persons name"
        print("Hello {}".format(name))
        return
```

Class Mehod: Greeting receiving a value for name

Output to server console

server_demo_3.py – gdbus tool testing method call

```
$ gdbus call --session --dest org.example.demo.test --object-path /org/example/
demo/test --method org.example.demo.test.greeting Bob
()
$ gdbus call --session --dest org.example.demo.test --obje                ple/
demo/test --method org.example.demo.test.greeting Fred
()
$ gdbus call --session --dest org.example.demo.test --object-path /org/example/
demo/test --method org.example.demo.test.greeting Wendy
()
$ gdbus call --session --dest org.example.demo.test --object-path /org/example/
demo/test --method org.example.demo.test.greeting "Paul Jones"
()
```

Testing Console

Receive empty tuple

```
$ python3 server_demo_3.py
Hello Bob
Hello Fred
Hello Wendy
Hello Paul Jones
```

Server Console

server_demo_4.py – Middle Section – Method Call send & receive data

```
class DBusService_XML():
    """
    DBus Service XML definition.
    type="i" for integer, "s" string, "d" double, "as" list of string data.
    """
    dbus = """
    <node>
        <interface name="{}">
            <method name="echo_string">
                <arg type="s" name="input" direction="in">
                </arg>
                <arg type="s" name="output" direction="out">
                </arg>
            </method>
        </interface>
    </node>
    """.format(BUS)

    def echo_string(self, input_string):
        "Echo the string"
        return input_string
```

Class Mehod: echo_string

Return the input received

server_demo_4.py – gdbus tool testing method call

```
$ python3 server_demo_4.py
```

Server Console

```
$ gdbus call --session --dest org.example.demo.test --object-path /org/example/
demo/test --method org.example.demo.test.echo_string "Hello World"

('Hello World',)

$ gdbus call --session --dest org.example.demo.test --object-path /org/example/
demo/test --method org.example.demo.test.echo_string "Once upon a time..."

('Once upon a time...',)

$ gdbus call --session --dest org.example.demo.test --object-path /org/example/
demo/test --method org.example.demo.test.echo_string "1234567890"

('1234567890',)
$
```

Receive tuple echoing string sent

Testing Console

Developing a Client application

.

Developing a client application. client_demo_1.py

```
# Importing
from pydbus import SessionBus  # from pydbus import SystemBus
from gi.repository import GLib

# Instantiation, Constants , Variables
bus = SessionBus()
BUS = "org.example.demo.test"
server_object = bus.get(BUS)
loop = GLib.MainLoop()
INTERVAL = 2

def make_method_call_client_1():
    print("Sending Method Call: server_no_args")
    reply = server_object.server_no_args()
    return True

if __name__=="__main__":
    print("Starting Client Demo 1...")
    # Call function to make a method call.
    GLib.timeout_add_seconds(interval=INTERVAL,
                             function=make_method_call_client_1)
    loop.run()
```

Instantiate using .get()

Method Call to server

GLib to run a repeating function every 2 seconds

Consoles: server_demo_1.py / client_demo_1.py

Start server before starting client

```
$ python3 server_demo_1.py
This is message 0
This is message 1
This is message 2
This is message 3
This is message 4
```

Server Console

```
$ python3 client_demo_1.py
Starting Client Demo 1...
Sending Method Call: server_no_args
Sending Method Call: server_no_args
Sending Method Call: server_no_args
Sending Method Call: server_no_args
Sending Method Call: server_no_args
```

Client Console

Every 2 seconds

Client_2 code and Consoles: server_demo_2.py / client_demo_2.py

```
def make_method_call_client_2():
    "Server returns a time stamp."
    reply = server_object.get_time_stamp()
    print("Returned data is of type: {}".format(type(reply)))
    print("Time stamp received from server: {}".format(reply))
    return True
```

Server returns data.
Client sends no data

```
$ python3 server_demo_2.py
```

Server Console

```
$ python3 client_demo_2.py
Starting...
Returned data is of type: <class 'float'>
Time stamp received from server: 1511690207.3769732
Returned data is of type: <class 'float'>
Time stamp received from server: 1511690209.375649
Returned data is of type: <class 'float'>
Time stamp received from server: 1511690211.3758538
```

Client Console

Client_3 code and Consoles: server_demo_3.py / client_demo_3.py

```
def make_method_call_client_3():
    "Client sends a random name to the server."
    name_list = ["Bob", "Pete", "Fred", "Sue", "Wendy"]
    name = name_list[random.randint(0, len(name_list)-1)]
    reply = server_object.greeting(name)
    return True
```

Client sends randomly selected "name" data

```
$ python3 server_demo_3.py
Hello Bob
Hello Fred
Hello Bob
Hello Wendy
Hello Sue
```

Server Console

```
$ python3 client_demo_3.py
Starting Client Demo 3...
```

Client Console

Client_4 code and Consoles: server_demo_4.py / client_demo_4.py

```
def make_method_call_client_4():
    name_list = ["Bob", "Pete", "Fred", "Sue", "Wendy"]
    name = name_list[random.randint(0, len(name_list)-1)]
    print("Name to be sent to server: {}".format(name))
    reply = server_object.echo_string(name)
    print("Message echoed from server: {}".format(reply))
    return True
```

Client sends randomly selected "name" data

Server replies with the name

```
$ python3 server_demo_4.py
```

Server Console

```
$ python3 client_demo_4.py
Starting Client Demo 4...
Name to be sent to server: Fred
Message echoed from server: Fred
Name to be sent to server: Wendy
Message echoed from server: Wendy
```

Client Console

Emit a Signal.

Server application emits a signal over the D-Bus:

server_demo_5.py – Beginning and End Sections.

```
# Importing...
from pydbus import SessionBus  # SystemBus
from pydbus.generic import signal
from gi.repository import GLib
import time
import random

# Variables / Constants / Instantiation...
bus = SessionBus()  # SystemBus
BUS = "org.example.demo.test"
loop = GLib.MainLoop()
INTERVAL = 2
message_count = 0
```

Add importing signal

```
if __name__ == "__main__":
    emit = DBusService_XML()
    bus.publish(BUS, emit)

    GLib.timeout_add_seconds(interval=INTERVAL, function=timer)
    loop.run()
```

Add instantiation

Glib to runs a repeating function

server_demo_5.py – Middle Section – Signal

```
class DBusService_XML():
    """
    DBus Service XML definition.
    type="i" for integer, "s" string, "d" double, "as" list of string data.
    """
    dbus = """
    <node>
        <interface name="{}">
            <signal name="integer_signal">
                <arg type="i"/>
            </signal>
        </interface>
    </node>
    """.format(BUS)
    integer_signal = signal()
```

Define integer_signal

Instantiate

```
def timer():
    "Emit a random integer each call."
    random_integer = random.randint(0, 100)
    print("Random integer emitted: {}".format(random_integer))
    emit.integer_signal(random_integer)
    return True
```

Emit signal

gdbus monitoring emitted signal.

```
$ python3 server_demo_5.py
Starting Server Demo 5...
Random integer emitted: 23
Random integer emitted: 5
Random integer emitted: 50
Random integer emitted: 84
Random integer emitted: 11
Random integer emitted: 43
```

Server Console

```
$ gdbus monitor --session --dest org.example.demo.testMonitoring signals from
all objects owned by org.example.demo.test
The name org.example.demo.test is owned by :1.173
/org/example/demo/test: org.example.demo.test.integer_signal (5,)
/org/example/demo/test: org.example.demo.test.integer_signal (50,)
/org/example/demo/test: org.example.demo.test.integer_signal (84,)
/org/example/demo/test: org.example.demo.test.integer_signal (11,)
/org/example/demo/test: org.example.demo.test.integer_signal (43,)
```

Testing Console

Client Subscribes to a Signal.

Client_5 code – Subscribe to emitted signals

```
from pydbus import SessionBus  # from pydbus import SystemBus
from gi.repository import GLib
# Instantiation, Constants, Variables...
bus = SessionBus()
BUS = "org.example.demo.test"
loop = GLib.MainLoop()

def cb_signal_emission(*args):
    "Callback on emitting signal, a random integer, from server. "
    # Data is in args[4]. The first item in a tuple. i.e. args[4][0]
    # print(args)
    random_number = args[4][0]
    print("Client received random number: {}".format(random_number))

if __name__=="__main__":
    print("Starting. Client Demo 5..")
    # Create the dbus filter based on: org.example.demo.test
    dbus_filter = "/" + "/".join(BUS.split("."))
    print(dbus_filter)
    # Subscribe to dbus to monitor for server signal emissions
    # dbus_filter. E.g. /org/example/demo/test
    bus.subscribe(object = dbus_filter, signal_fired = cb_signal_emission)
    loop.run()
```

Callback each time emitted data is received.

Create filter from BUS

.subscribe()

Consoles: server_demo_5.py / client_demo_5.py

```
$ python3 server_demo_5.py
Random integer emitted: 17
Random integer emitted: 78
Random integer emitted: 66
Random integer emitted: 96
Random integer emitted: 10
Random integer emitted: 59
Random integer emitted: 92
```

Server Console

```
$ python3 client_demo_5.py
Starting. Client Demo 5..
/org/example/demo/test
Client received random number: 78
Client received random number: 66
Client received random number: 96
Client received random number: 10
Client received random number: 59
Client received random number: 92
```

Client Console

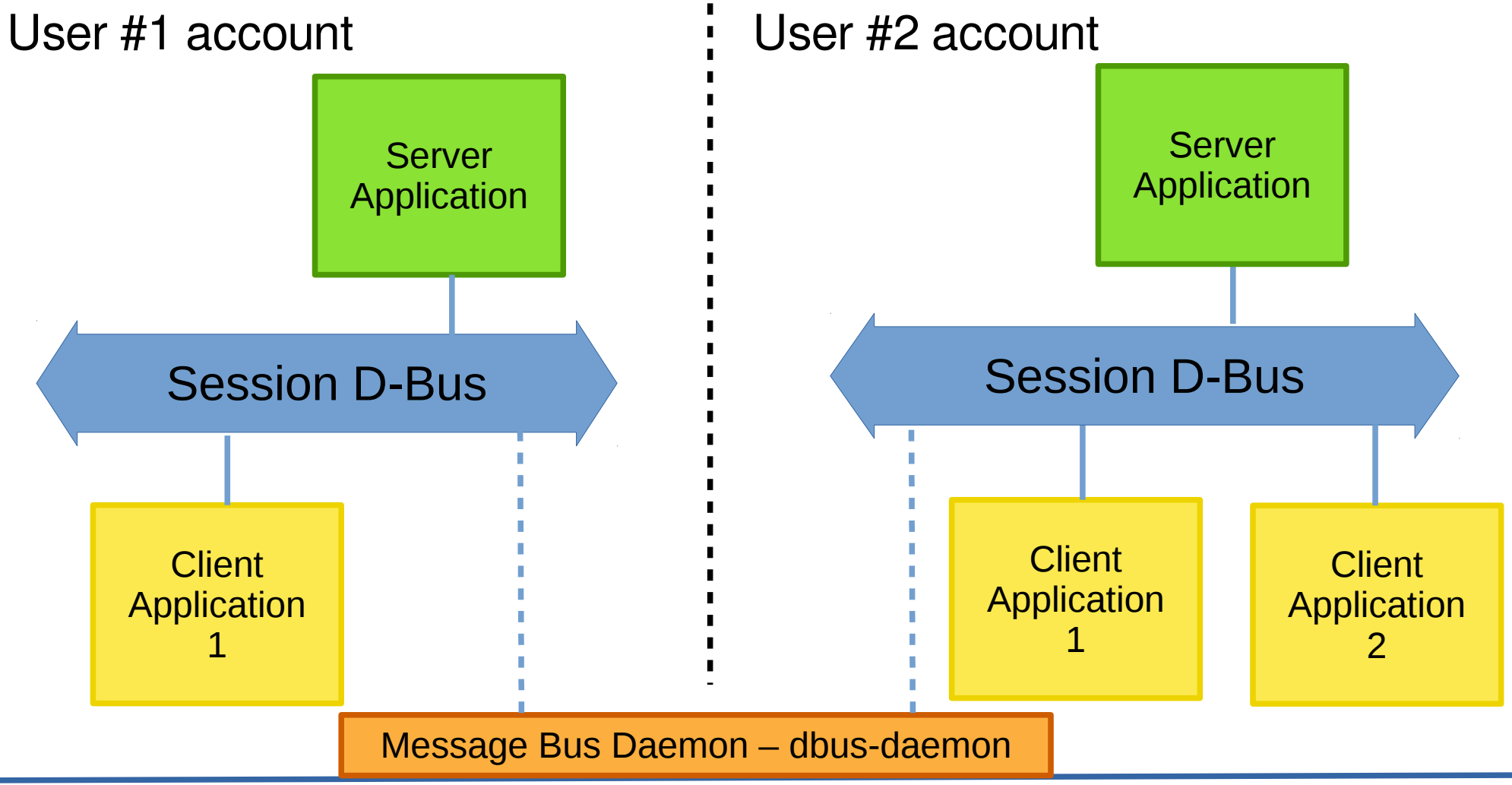
Perform Demo - server_demo_6 / client_demo_6

- Start server and demo gdbus receiving random numbers.
- Start client. 4 x method calls and subscribed to servers emitted signals.

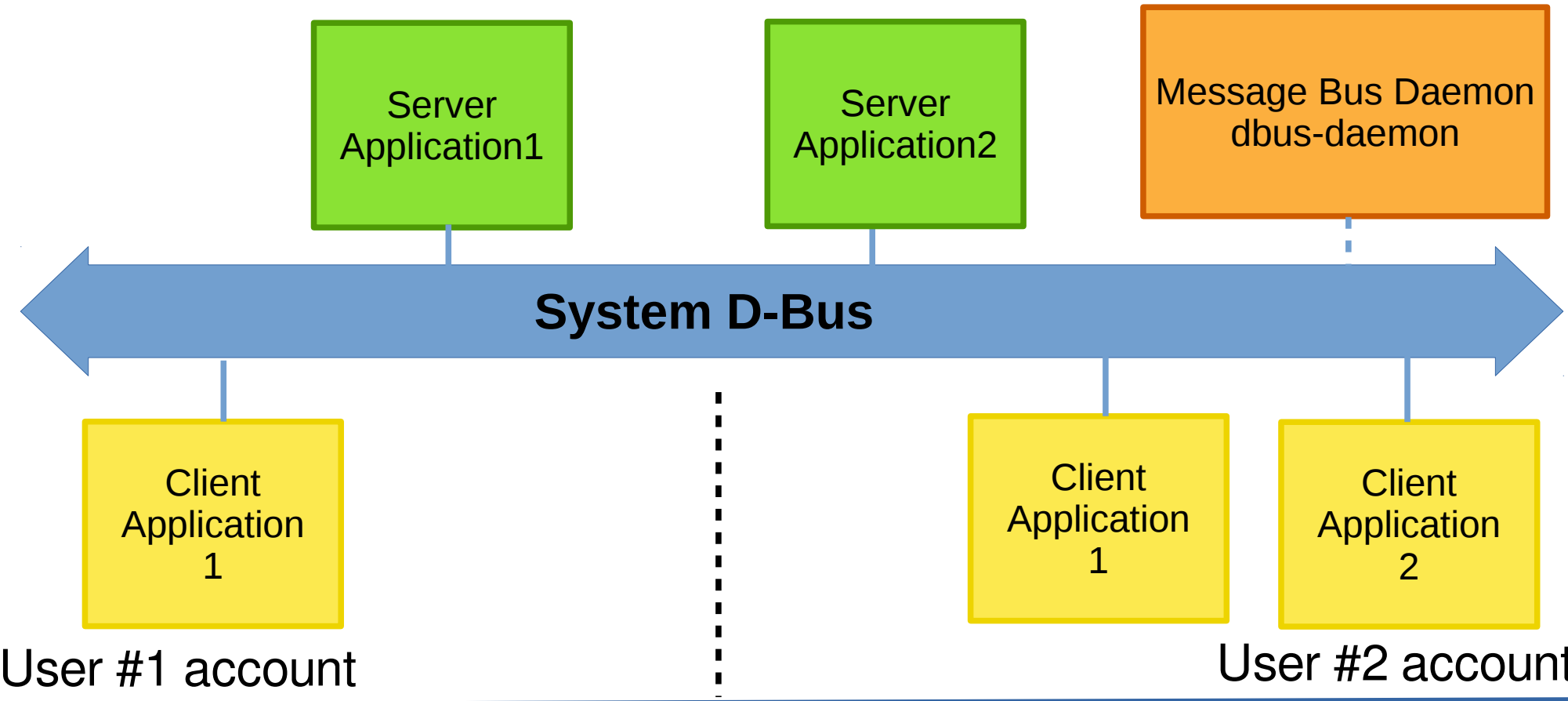
System D-Bus – rather than Session D-Bus

.

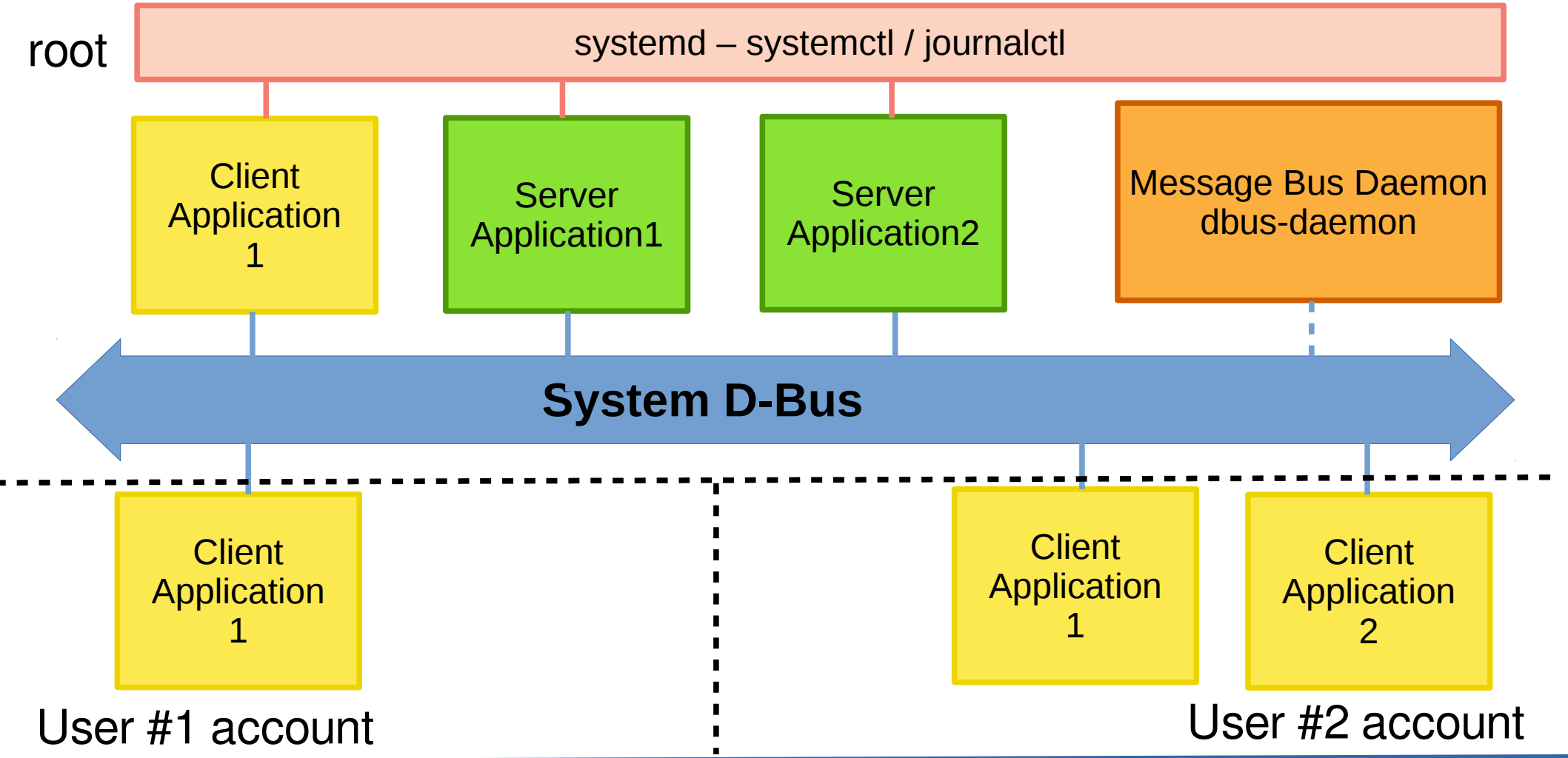
Client / Server applications. Session D-Bus



Client / Server applications. System D-Bus



Client / Server applications. System D-Bus. systemd control.



Setup and Demo System D-Bus

Create a suitable directory, then edit server code to use SystemBus...

```
$ pwd
/home/ian
$ mkdir demo
$ cd demo
$ mkdir test
$ cd test
$ pwd
/home/ian/demo/test
```

```
# Importing...
from pydbus import SystemBus  # SessionBus
from pydbus.generic import signal
from gi.repository import GLib
import time
import random

# Variables / Constants / Instantiation...
bus = SystemBus()  # SessionBus
BUS = "org.example.demo.test"
loop = GLib.MainLoop()
```

Use System D-Bus

Use System D-Bus

Error trying to run server...

```
$ python3 server_demo_7.py
Traceback (most recent call last):
... some error info deleted ...
GLib.Error: g-dbus-error-quark:
GDBus.Error:org.freedesktop.DBus.Error.AccessDenied: Connection ":1.150" is
not allowed to own the service "org.example.demo.test" due to security
policies in the configuration file (9)
```

/usr/share/dbus-1/system.conf file suggests adding a /etc/dbus-1/system-local.conf

```
$ ls /usr/share/dbus-1/
accessibility-services  services      system.conf  system-services
interfaces              session.conf  system.d

$ cat /usr/share/dbus-1/system.conf
<!-- This configuration file controls the systemwide message bus.
     Add a system-local.conf and edit that rather than changing this
     file directly. -->
... snip ...
  <!-- This is included last so local configuration can override what's
       in this standard file -->
  <include ignore_missing="yes">/etc/dbus-1/system-local.conf</include>
```

Add a system-local.conf

Looks for system-local.conf

Copy template from /usr/share/dbus-1/system.conf to /etc/dbus-1/system-local.conf

```
$ cd /etc/dbus-1/
$ ls
session.d  system.d
$ ls /usr/share/dbus-1/
accessibility-services  services      system.conf  system-services
interfaces              session.conf  system.d
$ sudo cp /usr/share/dbus-1/system.conf system-local.conf
[sudo] password for ian:
$ ls
session.d  system.d  system-local.conf
```

Edit system-local.conf to only contain <policy context=”default” section

```
<!DOCTYPE busconfig PUBLIC "-//freedesktop//DTD D-Bus Bus Configuration 1.0//EN"
 "http://www.freedesktop.org/standards/dbus/1.0/busconfig.dtd">
<busconfig>
  <policy context="default">
    <!-- All users can connect to system bus -->
    <allow user="*"/>

    <!-- Holes must be punched in service configuration files for
         name ownership and sending method calls -->
    <!-- Disable the deny...
    <deny own="*"/>
    <deny send_type="method_call"/> -->

    <!-- Change to allow. Ian - Nov 2017 -->
    <allow own="*"/>
    <allow send_type="method_call"/>

    <deny send_destination="org.freedesktop.DBus"
          send_interface="org.freedesktop.systemd1.Activator"/>
  </policy>
</busconfig>
```

Comment out the deny’s

Insert the allows

End of file. See this file in server_demo_7.py comments

After Reboot. Launch the server, and test it is emitting with gdbus-tool

```
$ ### AFTER REBOOTING ###
$ cd demo
$ cd test
$ ls
server_demo_7.py
$ python3 server_demo_7.py
Random integer emitted: 20
Random integer emitted: 27
Random integer emitted: 51
Random integer emitted: 58
Random integer emitted: 55
Random integer emitted: 47
```

Server Console

Now monitoring **system bus**

before was **session bus**

```
$ gdbus monitor --system --dest org.example.demo.test
Monitoring signals from all objects owned by org.example.demo.test
The name org.example.demo.test is owned by :1.53
/org/example/demo/test: org.example.demo.test.integer_signal (51,)
/org/example/demo/test: org.example.demo.test.integer_signal (58,)
/org/example/demo/test: org.example.demo.test.integer_signal (55,)
/org/example/demo/test: org.example.demo.test.integer_signal (47,)
```

Testing Console

Edit client_demo_7.py to use the SystemBus

Changed to SystemBus

```
# Importing...
from pydbus import SystemBus  # from pydbus import SessionBus
from gi.repository import GLib
import random

# Instantiation, Constants, Variables...
bus = SystemBus()
BUS = "org.example.demo.test"
```

SystemBus

Check Server and Client are using System D-Bus OK

```
$ python3 server_demo_7.py
Random integer emitted: 48
Random integer emitted: 26
Random integer emitted: 66
From Client 1: This is message 0
From Client 3: Hello Pete
Random integer emitted: 16
Random integer emitted: 16
From Client 1: This is message 1
From Client 3: Hello Fred
```

Server Console

```
$ python3 client_demo_7.py
Starting...
Emitted: Client received random number: 26
Emitted: Client received random number: 66
Sending Method Call: server_no_args
Client 2: Time stamp received from server: 1512125955.9466636
Client_4: Data Sent: H Echoed data: H
Emitted: Client received random number: 16
Emitted: Client received random number: 16
```

Client Console

Now get some systemd things running.
Files currently in /etc/systemd/system/...

```
$ cd /etc/systemd/system/
$ ls
bluetooth.target.wants                      graphical.target.wants
dbus-org.bluez.service                      multi-user.target.wants
dbus-org.freedesktop.Avahi.service          network-online.target.wants
dbus-org.freedesktop.ModemManager1.service  paths.target.wants
dbus-org.freedesktop.nm-dispatcher.service  printer.target.wants
dbus-org.freedesktop.resolve1.service       snap-core-3017.mount
dbus-org.freedesktop.thermald.service       snap-core-3440.mount
default.target.wants                        snap-pulsemixer-1.mount
display-manager.service                     sockets.target.wants
display-manager.service.wants               sysinit.target.wants
final.target.wants                          syslog.service
getty.target.wants                          timers.target.wants
```

Create /etc/systemd/system/server_demo.service

Open ▾ *server_demo.service /etc/systemd/system Save

```
1 # server_demo.service
2 # systemd service file for: ~/demo/test/server_demo_7.py
3 # Author: Ian Stewart
4 # Date: 2017-11-22
5 #
6 [Unit]
7 Description=server_demo_7.py is to demonstrate client/server using pydbus
8 After=multi-user.target
9
10 [Service]
11 Type=idle
12 ExecStart=/usr/bin/python3 /home/ian/demo/test/server_demo_7.py
13
14 [Install]
15 WantedBy=multi-user.target
```

Plain Text ▾ Tab Width: 4 ▾ Ln 15, Col 27 ▾ INS

Use systemctl to reload and start server_demo_7.py

```
$ sudo systemctl daemon-reload

$ sudo systemctl start server_demo.service

$ systemctl status server_demo
● server_demo.service - server_demo_7.py is to demonstrate client/server using p
   Loaded: loaded (/etc/systemd/system/server_demo.service; disabled; vendor pre
   Active: active (running) since Sat 2017-12-02 08:01:39 NZDT; 48s ago
 Main PID: 7061 (python3)
    Tasks: 3 (limit: 4915)
   CGroup: /system.slice/server_demo.service
           └─7061 /usr/bin/python3 /home/ian/demo/test/server_demo_7.py

Dec 02 08:01:39 ian-kiwi-pycon systemd[1]: Started server_demo_7.py is to demons
```

daemon-reload

start service to launch server_demo_7.py

Use systemctl “enable” for server_demo_7.py starts on re-boots...

```
$ sudo systemctl enable server_demo.service
Created symlink /etc/systemd/system/multi-user.target.wants/server_demo.service
→ /etc/systemd/system/server_demo.service.
```

Setup systemd

$ journalctl -e (trimmed). For debugging better to run server from bash

```
: Started Hostname Service.
: pam_unix(sudo:session): session closed for user root
:      ian : TTY=pts/0 ; PWD=/home/ian/demo/test : USER=root : COMMAND=/bin/
systemctl daemon-reload
: pam_unix(sudo:session): session opened f
: Reloading.
: pam_unix(sudo:session): session closed for user root
: Starting Daily apt download activities...
: [system] Activating via systemd: service name='org.freedesktop.PackageKit'
unit='packagekit.service'
: Starting PackageKit Daemon...
: daemon start
: [system] Successfully activated service 'org.freedesktop.PackageKit'
: Started PackageKit Daemon.
:      ian : TTY=pts/0 ; PWD=/home/ian/demo/test ; USER=root ; COMMAND=/bin/
systemctl start server_demo.service
: pam_unix(sudo:session): session opened for user root by (uid=0)
: Started server_demo_7.py is to demonstrate client/server using pydbus.
: pam_unix(sudo:session): session closed for us
: Random integer emitted: 63
: Random integer emitted: 8
```

Reload systemd manager configuration

Start the server

print() goes into the log.
Probably don't want this.

Notes:

- Client application may also be started by systemctl
- Alternative is to insert into server code a subprocess.call() to start client.
- Having the code in a User account ~/demo/test is easier to troubleshoot / edit the code.
- Final code version could be moved.
- Files created by server app, will be owned by root when created after launching with systemctl. May need some chown().

Demo – with GUI clients.

Example with Cycle Analyst.

Any Questions?

Note:

- All demo code and presentation material is available at: https://github.com/irsbugs/kiwipycon2017

End.

Thank you.