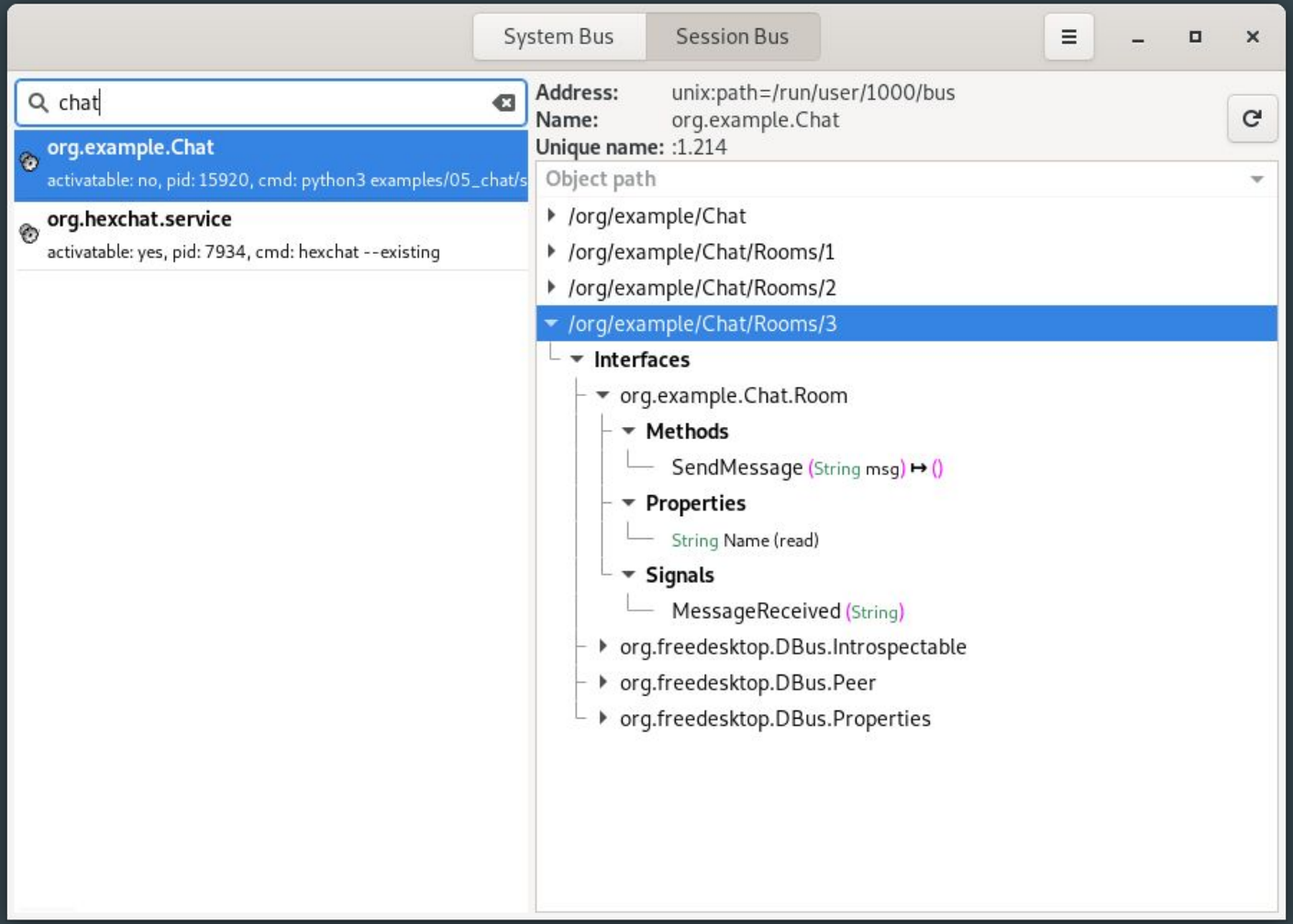
Talk to D-Bus from a Python application

Vendula Poncova
Brussels, 2023

D-Bus

protocol - bus daemon - inter-process


System Bus
Session Bus
chat
org.example.Chat
activatable: no, pid: 15920, cmd: python3 examples/05_chat/s
org.hexchat.service
activatable: yes, pid: 7934, cmd: hexchat --existing
Address: unix:path=/run/user/1000/bus
Name: org.example.Chat
Unique name: :1.214
Object path
/org/example/Chat
/org/example/Chat/Rooms/1
/org/example/Chat/Rooms/2
/org/example/Chat/Rooms/3
Interfaces
org.example.Chat.Room
Methods
SendMessage (String msg) ↦ ()
Properties
String Name (read)
Signals
MessageReceived (String)
org.freedesktop.DBus.Introspectable
org.freedesktop.DBus.Peer
org.freedesktop.DBus.Properties

dasbus library

Python - GLib - D-Bus

Part 1: Client

Create a D-Bus proxy

Specify the D-Bus name of the service and D-Bus path of the object.

```
bus = SessionMessageBus()
proxy = bus.get_proxy(
    "org.example.Chat",
    "/org/example/Chat/Rooms/3"
)
```

Get a D-Bus property

Get a property of the D-Bus proxy.

```
print(proxy.Name)
```

Call a D-Bus method

Call a method of the D-Bus proxy.

```
proxy.SendMessage("Hello World!")
```

Watch a D-Bus signal

Connect a callback to the signal of the D-Bus proxy and start the event loop.

```
def callback(message):
    print(message)

proxy.MessageReceived.connect(
    callback
)
```

Run the event loop

Process asynchronous communication.

```
loop = EventLoop()
loop.run()
```

Part 2: Service

Register a D-Bus service

Specify a D-Bus name of the service.

```
bus = SessionMessageBus()
bus.register_service(
    "org.example.Chat"
)
```

Publish a D-Bus object

Specify the D-Bus path of the object.

```
bus.publish_object(
    "/org/example/Chat/Rooms/3",
    Room()
)
```

Start the event loop

Start to handle incoming D-Bus calls.

```
loop = EventLoop()
loop.run()
```

Part 3: Object

XML Specification

```
<?xml version="1.0" ?>
<node>
  <interface name="org.example.Chat.Room">
   <method name="SendMessage">
      <arg direction="in" type="s"/>
   </method>
  </interface>
</node>
```

Generate the XML

Use a decorator to generate the XML.

```
@dbus_interface("org.example.Chat.Room")
class Room(object):
```

Define a D-Bus method

Use type hints to define types.

```
def SendMessage(self, msg: Str):
    print(msg)
```

Define a D-Bus property

Use type hints to define types.

```
@property
def Name(self) -> Str:
    return "Room 3"
```

Define a D-Bus signal

Specify the arguments and types.

```
@dbus_signal
def MessageReceived(self, msg: Str):
    pass
```

Part 4: Features

Manage D-Bus names

Pre-define D-Bus names and paths of services, objects and interfaces.

```
proxy = CHAT.get_proxy(ROOM_3)
```

Generate D-Bus paths

Manage a group of publishable objects.

```
path = ROOM_CONTAINER.to_object_path(
    room
)
```

Handle D-Bus errors

Map names of DBus errors to exceptions.

```
try:
    proxy.SendMessage(...)
except ChatError:
    ...
```

Handle D-Bus structures

Represent dictionaries of variants by Python objects.

```
data = UserData.from_structure(
    structure
)
print(data.name)
```

Use Unix file descriptors

Send and receive Unix file descriptors.

```
proxy = bus.get_proxy(
    ...
    client=GLibClientUnix
)
```

I don't need X

Features are optional.

I need Z

Classes are replaceable.

https://github.com/rhinstaller/dasbus