

Komunikacja międzyprocesowa z D-Busem

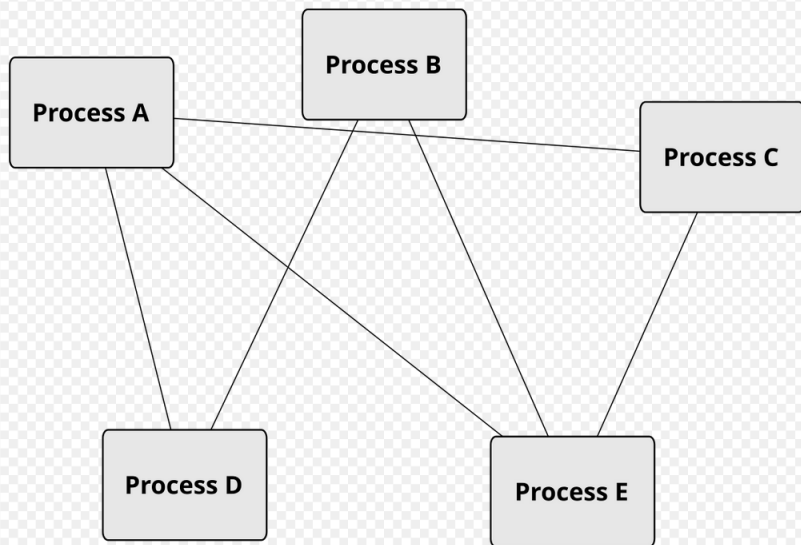
Maksymilian Wnuk

25 listopada 2024

Plan

- Historia
- Teoria
- Praktyka

Komunikacja międzyprocesowa



Przykłady

- Pamięć współdzielona
- Sockety
- Pipe, named pipe

Wysokopoziomowe rozwiązania

- CORBA, Common Object Request Broker Architecture
 - Skomplikowany, złożony standard
 - Transparentność lokacji
 - Problemy z restrykcyjnym firewallem
 - Brak planu na standard

Wysokopoziomowe rozwiązania

- CORBA, Common Object Request Broker Architecture
 - Skomplikowany, złożony standard
 - Transparentność lokacji
 - Problemy z restrykcyjnym firewallem
 - Brak planu na standard
- DCOP, Desktop COmmunication Protocol
 - Daemon, implementacja klient-serwer z traffic directorem
 - C/C++
 - Zastąpiony przez D-Bus

Projekt z 2002 roku zarządzający między innymi:

- PulseAudio
- systemd
- Wayland
- Mesa

D-Bus

Potrzeba zaimplementowania ustandaryzowanego, bezpiecznego IPC dla środowisk graficznych.

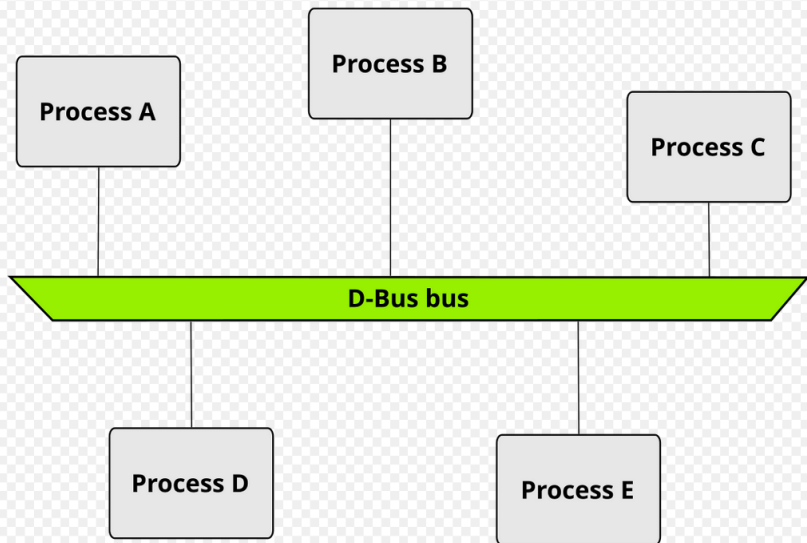
2002 - początek projektu

2006 - stabilna wersja

Zalety

- Prosty
- Dostępność standardu w wielu językach programowania
- Security policy
- Abstrakcja połączenia
- Walidacja typów
- Centralizacja
- Działanie ad hoc

Działanie



D-Bus bus

Demon do którego łączą się aplikacje. Przekierowuje wiadomości od aplikacji do innych aplikacji. Podział na system bus i session bus.

Session bus

Z poziomu użytkownika, osobna instancja dla każdego.

- DE KDE, GNOME
- Aplikacje, Spotify, Firefox
- PulseAudio

System bus

Z poziomu systemu operacyjnego. Pojedyncza instancja dla każdego użytkownika.

- Sieci, NetworkManager
- Urządzenia, UDisks
- Uprawnienia, PolicyKit

Połączenie serwisu

Następuje przy połączeniu do demonu. Serwis ma przyznaną nazwę.

- Unikalna: :1-37, :1-42
- Well-known name: org.Cinammon, org.mpris.MediaPlayer2.spotify

Demon mapuje well-known name na nazwę unikalną.

Coś jak dns.

Interfejsy

Zbiór metod, sygnałów oraz właściwości (properties). Implementowane przez obiekty. Coś jak abstract class.

- `org.mpris.MediaPlayer2.Player`
- `org.freedesktop.Notifications`
- `org.freedesktop.NetworkManager`
- `org.freedesktop.UDisks2.Drive`

Interfejsy standardowe

Interfejsy zapewniające definicje metod, przydatne przy implementacji serwisów. Bindingi mogą implicity implementować!

- `org.freedesktop.DBus.Peer` - metody `Ping` i `GetMachineId`

Interfejsy standardowe

Interfejsy zapewniające definicje metod, przydatne przy implementacji serwisów. Bindingi mogą implicity implementować!

- `org.freedesktop.DBus.Peer` - metody `Ping` i `GetMachineId`
- `org.freedesktop.DBus.Introspectable` - reprezentacja obiektu w XML

Interfejsy standardowe

Interfejsy zapewniające definicje metod, przydatne przy implementacji serwisów. Bindingi mogą implicity implementować!

- `org.freedesktop.DBus.Peer` - metody `Ping` i `GetMachineId`
- `org.freedesktop.DBus.Introspectable` - reprezentacja obiektu w XML
- `org.freedesktop.DBus.Properties` - manipulacja właściwościami, odczyt wartości

Interfejsy standardowe

Interfejsy zapewniające definicje metod, przydatne przy implementacji serwisów. Bindingi mogą implicity implementować!

- `org.freedesktop.DBus.Peer` - metody `Ping` i `GetMachineId`
- `org.freedesktop.DBus.Introspectable` - reprezentacja obiektu w XML
- `org.freedesktop.DBus.Properties` - manipulacja właściwościami, odczyt wartości
- `org.freedesktop.DBus.ObjectManager` - zarządzanie obiektami udostępnianymi przez serwis

Obiekt

- Wymaga określenia bus name
- Ścieżka do obiektu jako nazwa:
 - /org/mpris/MediaPlayer2
 - /org/freedesktop/Notifications
 - /org/freedesktop/NetworkManager
 - /org/freedesktop/UDisks2/drives/KINGSTON123afmaolk1
- Implementuje interfejs lub interfejsy.

Metody

Synchroniczne. Obiekt implementuje metodę którą klient może wywołać z argumentami. Może zwracać z powrotem wartość.

Sygnały

Asynchroniczne. Obiekt emituje sygnał. Klient nasłuchuje sygnałów i implementuje handler.

Przykład

```

Address:    unix:path=/run/user/1000/bus
Name:      org.mpris.MediaPlayer2.spotify
Unique name: :1.255

Object path
└─ Match rules
└─ Statistics
└─ /org/mpris/MediaPlayer2
   └─ Interfaces
      ├── org.freedesktop.DBus.Introspectable
      ├── org.freedesktop.DBus.Peer
      ├── org.freedesktop.DBus.Properties
      ├── org.mpris.MediaPlayer2
      └─ org.mpris.MediaPlayer2.Player
         └─ Methods
            ├── Next () ⇨ ()
            ├── OpenUri (String Uri) ⇨ ()
            ├── Pause () ⇨ ()
            ├── Play () ⇨ ()
            ├── PlayPause () ⇨ ()
            ├── Previous () ⇨ ()
            ├── Seek (Int64 Offset) ⇨ ()
            ├── SetPosition (Object Path TrackId, Int64 Position) ⇨ ()
            └─ Stop () ⇨ ()

```

```

└─ Properties
   ├── Boolean CanControl (read)
   ├── Boolean CanGoNext (read)
   ├── Boolean CanGoPrevious (read)
   ├── Boolean CanPause (read)
   ├── Boolean CanPlay (read)
   ├── Boolean CanSeek (read)
   ├── Boolean Shuffle (read / write)
   ├── Dict of (String, Variant) Metadata (read)
   ├── Double MaximumRate (read)
   ├── Double MinimumRate (read)
   ├── Double Rate (read / write)
   ├── Double Volume (read / write)
   ├── Int64 Position (read)
   ├── String LoopStatus (read / write)
   └─ String PlaybackStatus (read)

└─ Signals
   └─ Seeked (Int64)

```

Proxy

Obiekt tworzony przez klienta jako reprezentacja zdalnego obiektu innego procesu. Niskopoziomowo tworzy wiadomość do serwisu z requestowanym wykonaniem metod i zwraca odpowiedź.

```
bus = SessionBus()  
proxy = bus.get(SERVICE_NAME, SERVICE_OBJECT)  
method_res = proxy[INTERFACE_NAME].Method(a1, a2)  
  
proxy.SomeSignal().connect(SIGNAL_HANDLER)  
loop.run()
```


Wiadomości niskopoziomowo

Dzieli się na 4 typy:

- Wiadomości wywołujące metody
- Wiadomości zwracające return value metody
- Błędy zwracające wyjątki spowodowane wywołaniem metody. Zła walidacja typu, brak autoryzacji, brak serwisu, obiektu itp.
- Asynchroniczne sygnały

Wiadomości niskopoziomowo cd.

Coś takiego:

```
Message message = new Message("/remote/object/path", "MethodName", arg1, arg2);
Connection connection = getBusConnection();
connection.send(message);
Message reply = connection.waitForReply(message);
if (reply.isError()) {

} else {
    Object returnValue = reply.getReturnValue();
}
```

System typów

Parę przykładów:

- aai - tablica tablic intów

System typów

Parę przykładów:

- `aai` - tablica tablic intów
- `a(ss)` - array structów z dwoma stringami

System typów

Parę przykładów:

- `aai` - tablica tablic intów
- `a(ss)` - array structów z dwoma stringami
- `a{sd}` - mapa klucz-string value-double

System typów

Parę przykładów:

- `aai` - tablica tablic intów
- `a(ss)` - array structów z dwoma stringami
- `a{sd}` - mapa klucz-string value-double
- `v` - variant, dynamiczny typ opakowany w klasę `Variant`

System typów

Parę przykładów:

- `aai` - tablica tablic intów
- `a(ss)` - array structów z dwoma stringami
- `a{sd}` - mapa klucz-string value-double
- `v` - variant, dynamiczny typ opakowany w klasę `Variant`
- `a{sv}` - mapa z wartościami o różnych typach

System typów

Parę przykładów:

- `aai` - tablica tablic intów
- `a(ss)` - array structów z dwoma stringami
- `a{sd}` - mapa klucz-string value-double
- `v` - variant, dynamiczny typ opakowany w klasę `Variant`
- `a{sv}` - mapa z wartościami o różnych typach
- Ale tak naprawdę wszystko jest `variantem` (enkapsulacja)

XML

```
<node>
  <interface name="org.meks.Logger">
    <method name="AddLog">
      <arg direction="in" type="s" name="log_str" />
      <arg direction="out" type="s" />
    </method>
    <signal name="CountChange">
      <arg type="(ib)" name="count" />
    </signal>
    <property name="Upper" type="b" access="readwrite"/>
    <property name="LogCount" type="i" access="read" />
    <property name="LogFileName" type="s" access="readwrite"/>
  </interface>
</node>
```

Policy XML

Kontrola dostępu do serwisów, metod i sygnałów. Znajduje się w
`/etc/dbus-1/system.d` lub `/usr/share/dbus-1/system.d`

Przykład: `NetworkManager.conf`

Narzędzia

- `dbus-monitor` - debugger do monitorowania wiadomości dbusa
- `dbus-send` - wysyłanie wiadomości do busy
- `d-feet` - graficzny debugger do sprawdzania serwisów i wysyłania wiadomości
- `busctl` - komenda do badania serwisów
- `gdbus-codegen` - generowanie kodu w C, biblioteka GDBus

Implementacje, Bindingi

Freedesktop.org oferuje implementacje i bindingi protokołu w D-Bus.
Najciekawsze:

- libdbus - niskopoziomowe API w C, nie polecane do pisania prostych aplikacji
- GDBus - niezależna od libdbus implementacja d-bus w C
- pydbus - wysokopoziomowy wrapper w pythonie na podstawie GDBus
- zbus, DBus-Java i inne

Podsumowanie

- IPC służące do wykonywania metod i nasłuchiwanie sygnałów serwisów
- Serwisy składające się z obiektów implementujących interfejsy z definicjami sygnałów, metod i właściwości(properties)
- Dobre do wymiany pojedynczych wiadomości, kontrola
- Fatalne do wymiany danych, szczególnie dużych
- Walidacja typów i autoryzacja dzięki security policy

Źródła

- <https://dbus.freedesktop.org/doc/dbus-tutorial.html>
- <https://dbus.freedesktop.org/doc/dbus-specification.html>
- <https://www.freedesktop.org/wiki/IntroductionToDBus>
- <https://dbus.freedesktop.org/doc/dbus-python>
- <https://en.wikipedia.org/wiki/D-Bus>
- <https://news.ycombinator.com/item?id=8648437>