D-BUS

Carlos Garcia Campos
carlosgc@gnome.org

Vilanova i la Geltrú

24 de junio de 2006

Que es D-BUS?

- Sistema de paso de mensajes que permite a las aplicaciones comunicarse entre ellas de una manera simple
- Mecanismo IPC (Inter-Process Comunication)
- Diseñado para cubrir dos casos de uso bien definidos:
  - Un bus de sistema: para la comunicación entre el sistema y las diferentes sesiones de usuario. Por ejemplo, notificaciones de cambios en el hardware.
  - Un bus de sesión: para la comunicación entre aplicaciones de escritorio.
- D-BUS ha sido diseñado por y para el escritorio

Arquitectura D-BUS

- D-BUS está dividido fundamentalmente en 3 capas
  - La librería libdbus: proporciona un API para que dos aplicaciones se comuniquen entre si intercambiando mensajes. Las comunicaciones son siempre 1 a 1.
  - El bus de mensajes: es un demonio creado sobre libdbus al que las aplicaciones pueden conectar. Es el encargado de administrar los mensajes, redirigiéndolos al destinatario o destinatarios.
  - Bindings: proporcionan un API de mas alto nivel que libdbus para distintos lenguajes de programación o frameworks de desarrollo abstrayendo al programador de los detalles de bajo nivel. Actualmente existen bindings para GLib/GObject, Qt, Python, Perl, Mono/C# y Java, aunque no todos ellos están incluidos de forma oficial en el paquete D-BUS.

Objetos

- Cuando una aplicación envía mensajes a otra lo hace siempre a un objeto concreto, no a toda la aplicación
- Las aplicaciones D-BUS tendrán siempre al menos un objeto
- Son referenciados a través de un path (de forma similar a un fichero en un sistema de ficheros)
- Path en D-BUS equivale a puntero o referencia en POO.
- Los objetos son instancias, no tipos
- Podemos invocar métodos sobre los objetos

Métodos y señales

- Son tipos de mensajes que pueden viajar por el bus.
- Invocar un método implica enviarle un mensaje de tipo método al objeto.
- Las señales son mensajes que normalmente no tienen destinatario. El bus redigirá estos mensajes todos los clientes conectados.

Métodos y señales

- Por el bus de mensajes solo pueden circular 4 tipos de mensajes:
  - Llamadas a métodos
  - Retorno de métodos
  - Mensaje de error (excepciones)
  - Señales

Interfaces

- Qué métodos pueden ser invocados sobre cada objeto viene definido por su interfaz.
- Un objeto puede implementar varias interfaces
- Las interfaces si que representan el tipo de un objeto
- Se representan en forma de nombre de dominio inverso. org.freedesktop.InterfaceName

Servicios

- Objetos, mensajes e interfaces se encuentran al nivel de la primera capa de la arquitectura (libdbus)
- Los servicios se encuentran al nivel de la segunda capa (el bus de mensajes)
- No son mas que un nombre conocido en un bus de mensajes. El bus de mensajes no sabe en realidad nada de servicios, sino de nombres que identifican clientes
- Un servicio es un conjunto de interfaces proporcionado por una aplicación bajo un nombre. Una aplicación puede proveer uno o mas servicios.
- Cuando una aplicación quiere publicar un servicio se conecta al bus de mensajes proporcionando un nombre de servicio que será el que usen los clientes para poder usar los objetos del servicio.

Servicios

- Un servicio, no existe como tal, es solo una forma de llamar a una aplicación que ofrece cierta funcionalidad a través de un nombre
- Este nombre se especifica igual que las interfaces, a través del sistema de nombres de dominio inverso.

El servicio calculadora

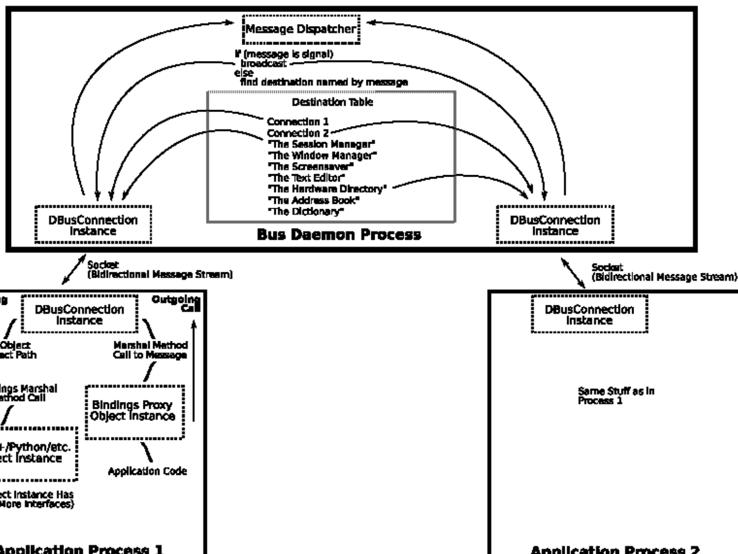
- En función de lo visto hasta ahora, podríamos convertir la calculadora de GNOME en un servicio D-BUS proporcionando:
  - Un nombre de servicio: org.gnome.utils.Calculator
  - Una interfaz (al menos): org.gnome.utils.Calculator
  - Los métodos de la interfaz: org.gnome.utils.Calculator.Add, org.gnome.utils.Calculator.Substract, etc.
  - Un objeto (al menos): /org/gnome/utils/Calculator

El bus de mensajes

- Es la segunda de la arquitectura construida sobre libdbus
- libdbus proporciona una comunicación 1 a 1, lo cual no parece muy últil si queremos mantener varias aplicaciones comunicadas entre si o realizar una notificación a todas las aplicaciones del ecritorio
- El bus de mensajes es en realidad un servicio mas, es decir, es uno de los participantes en la comunicación 1 a 1.
- Cada una de las aplicaciones o servicios completan de forma independiente el segundo participante de la comunicación.
- Cuando una aplicación quiere comunicarse con otra, lo que hace es enviarle sus mensajes al bus de mensajes y éste reenviará dichos mensajes al destinatario correspondiente

El bus de mensajes

- Cuando un cliente se conecta al bus de mensajes éste genera un nombre especial para él llamado nombre de conexión único
- Estos nombres especiales empiezan siempre por ':' y son creados dinámicamente de forma que nunca se repiten en una misma sesión del bus de mensajes (:1.3)
- Además del nombre único las aplicaciones pueden pedir al bus de mensajes un nombre bien conocido o nombre de servicio. Por ejemplo org.gnome.utils.Calculator


Message Dispatcher
if (message is signal)
broadcast
else
find destination named by message
Destination Table
Connection 1
Connection 2
"The Session Manager"
"The Window Manager"
"The Screensaver"
"The Text Editor"
"The Hardware Directory"
"The Address Book"
"The Dictionary"
DBusConnection Instance
Bus Daemon Process
DBusConnection Instance
Socket (Bidirectional Message Stream)
Socket (Bidirectional Message Stream)
Incoming Call
DBusConnection Instance
Outgoing Call
Locate Object via Object Path
Marshal Method Call to Message
Bindings Marshal to Method Call
Bindings Proxy Object Instance
C/C++/Python/etc. Object Instance
Application Code
(Object Instance Has 1 or More Interfaces)
Application Process 1
DBusConnection Instance
Same Stuff as in Process 1
Application Process 2

Cliente y servidor

- La comunicación 1 a 1 ofrecida por libdbus sigue el modelo tradicional cliente-servidor.
- Servidor: permanecerá la espera de conexiones
- Cliente: conecte al servidor
- Una vez establecida la conexión comienza una comunicación bidireccional de igual a igual, y libdbus ya no diferenciará entre cliente y servidor.
- Como en toda comunicación cliente-servidor, ambos deben ponerse de acuerdo en donde escuchará el servidor para que el cliente pueda conectar. En D-BUS se conoce como dirección D-BUS y por defecto es un socket unix.

Cliente y servidor

- El bus de mensajes es quién juega el papel del servidor y todas las demás aplicaciones que conectan al bus son clientes
- Para que las aplicaciones no tengan que ponerse de acuerdo con el bus de mensajes en cuanto a la dirección D-BUS, el bus de mensajes exporta en una variable de entorno dicha dirección.
- Libdbus consulta esa variable cuando un cliente trata de conectar a un servidor
- **Servicio != Servidor**

El servicio org.freedesktop.DBus

- Las aplicaciones pueden conectar al bus de mensajes para comunicarse con otras, pero también para interactuar con el propio bus de mensajes.
- Este tipo de interacciones con el bus de mensajes se realiza a través del servicio org.freedesktop.DBus ofrecido por el propio bus
- El servicio org.freedesktop.DBus ofrece un objeto /org/freedesktop/DBus que implementa la interfaz org.freedesktop.DBus

Ejemplo: ListNames

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import dbus

# Primero conectamos al bus de mensajes de sesión
bus = dbus.SessionBus ()

# Obtenemos el objeto /org/freedesktop/DBus
# proporcionado por el servicio org.freedesktop.DBus
remote_object = bus.get_object (``org.freedesktop.DBus'',
``/org/freedesktop/DBus'')

# Puesto que un objeto puede implementar varias
# interfaces, seleccionamos la interfaz a través de la
# cual vamos a interactuar con el objeto, es decir, la
# que ofrece el metodo ListNames en este caso
iface = dbus.Interface (remote_object, ``org.freedesktop.DBus'')

# Ahora invocamos el metodo ListNames para obtener la lista de nombre
names = remote_object.ListNames ()

# Mostramos los nombres
for name in names:
    print name
```

Servicios D-BUS

- Para crear un servicio simplemente tenemos que pedirle al bus un nombre
- Pedir un nombre es una de las tareas administrativas que es posible realizar a través del servicio org.freedesktop.DBus
- A diferencia del nombre de conexión único, el nombre de servicio requerido al bus si que puede variar a lo largo de una sesión de manera que distintas aplicaciones podrían poseer el mismo nombre en momentos distintos de la misma sesión.

Creación del servicio calculadora

- Conectamos al bus de mensajes y obtenemos un número de conexión único
- Obtenemos una referencia al objeto /org/freedesktop/DBus proporcionado por el servicio org.freedesktop.DBus
- Invocamos el método RequestName a través de la interfaz org.freedesktop.DBus sobre el objeto /org/freedesktop/DBus. Como argumento le pasamos el nombre org.gnome.utils.Calculator
- Si todo va bien el bus de mensajes asocia nuestra aplicación, a través del número de conexión único, al nombre de servicio org.gnome.utils.Calculator
- Por último publicamos una ruta al objeto /org/gnome/utils/Calculator en el bus

Estandarización

- Puede parecer un tanto ilógico que dos aplicaciones quieran el mismo nombre de servicio, pero vamos a verlo con un ejemplo.
- Actualmente en GNOME tenemos disponibles mas de 5 reproductores de música. Si cada uno quisiera proporcionar un servicio D-BUS es posible que todos quieran ser el servicio org.gnome.MusicPlayer.
- Normalmente solo uno de ellos está reproduciendo música a la vez así que podríamos programar un cliente que siempre acceda al servicio org.gnome.MusicPlayer sin importarle que aplicación es la que está ejecutando en ese momento.
- Si todos los servicios publican el mismo interfaz el uso sel servicio será totalmente independiente de la aplicación.

Estandarización

- Gracias a esta estandarización programar un applet que controle el reproductor de música es trivial.
- El código del applet será mucho mas limpio y mantenible, ya que será el mismo para soportar todos los resproductores de música.
- Si siguen apareciendo nuevos reproductores, serán también soportados directamente sin necesidad de tocar el código del applet o volver a compilarlo.
- El único requisito imprescindible para que esto sea así, es la estandarización. Siendo aún mas genéricos, podemos proporcionar un servicio org.freedesktop.MusicPlayer y el applet funcionará en cualquier sistema de escritorio que siga el estándar.

El método org.freedesktop.DBus.RequestName en detalle

- Puesto que un mismo nombre de servicio puede ser requerido por varias aplicaciones, el proceso de adquisición de nombres es relativamente complejo.
- ¿Qué pasa cuando dos aplicaciones quiere el mismo nombre de servicio al mismo tiempo? La respuesta depende los flags indicados en la llamada al método RequestName
- Estos flags son:
  - DBUS_NAME_FLAG_ALLOW_REPLACEMENT
  - DBUS_NAME_FLAG_REPLACE_EXISTING
  - DBUS_NAME_FLAG_DO_NOT_QUEUE

El método org.freedesktop.DBus.RequestName en detalle

- Si A tiene establecido DBUS_NAME_FLAG_ALLOW_REPLACEMENT y B ha invocado el método con el flag DBUS_NAME_FLAG_REPLACE_EXISTING, entonces B pasa a ser el "dueño" del nombre ocupando la cabeza de la cola y A pasa a la segunda posición de la cola. Si B ya estaba en la cola, sus flags serán actualizados además de saltar a la cabeza de la cola. Ejemplo: queremos reemplazar la versión actual de un servicio por otra mas moderna sin dejar de ofrecer dicho servicio.
- Si A no ha establecido DBUS_NAME_FLAG_ALLOW_REPLACEMENT o B no ha invocado el método con el flag DBUS_NAME_FLAG_REPLACE_EXISTING, A continuará siendo el poseedor del nombre y B pasará a la cola. Ejemplo: estamos usando un reproductor de música, pero decidimos dejar de usar este y empezar a usar otro. El applet de control del reproductor no notará el cambio de servicio.
- Si B ha invocado el método con el flag DBUS_NAME_FLAG_DO_NOT_QUEUE y no consigue ser el poseedor del nombre, la llamada fallará y en ningún caso B pasará a la cola. Ejemplo: queremos una aplicación de tipo "single instance" cuando detectamos que ya existe una instancia de nuestro servicio simplemente terminamos.

Iniciación de servicios

- El bus de mensajes es capaz de iniciar servicios bajo demanda
- Si recibe un mensaje para un cliente que no está conectado al bus, lo iniciará, esperará a que dicho cliente conecte y le enviará el mensaje
- Lógicamente para que esto suceda, el bus de mensajes tiene que tener conocimiento de como iniciar dicho servicio. Para ello se utilizan los ficheros de descripción de servicios
- Los ficheros de descripción de servicios son sencillos ficheros de tipo INI que contienen una sección [D-BUS Service] con la información necesaria para iniciar un servicio

Fichero de descripción de servicio

```
[D-BUS Service]
Name=org.gnome.utils.Calculator
Exec=/usr/bin/gnome-calculator
```

- Para que el bus de mensajes sea capaz de encontrarlo e identificarlo como un fichero de descripción de servicios debe cumplir los siguientes requisitos:
  - Estar en una ruta específica (por defecto /usr/share/dbus-1/services)
  - Tener extensión .service
  - El nombre del fichero debe coincidir con el nombre de servicio.
- /usr/share/dbus-1/services/org.gnome.utils.Calculator.service

Bindings

- Los bindings representan la tercera capa de la arquitectura de D-BUS
- Es la capa que facilita al programador la tarea de escribir aplicaciones D-BUS
- Gracias a los bindings el API de D-BUS es adaptado a cada lenguaje de programación o plataforma de desarrollo
- Es posible prescindir de ésta última capa y utilizar únicamente libdbus
- Los bindings hacen el sistema mas abierto y accesible, ya que nos permite programar con el sistema con el que mas cómodos nos sentimos, usar los tipos de datos dicho sistema, los objetos, etc.

D-BUS en el escritorio de hoy

- CORBA ha sido el estandar utilizado tanto en KDE como en GNOME.
- KDE pronto abandonó CORBA y desarrollaron un sistema mucho mas simple al que llamaron DCOP, usado hasta ahora con bastante éxito.
- En GNOME se desarrolló bonobo manteniendo CORBA como sistema IPC.
- D-BUS surge con unos objetivos muy similares a DCOP (siguiente generación de DCOP) disponible para los distintos sistemas de escritorio libres que siguen el estandar freedesktop.org.

D-BUS en el escritorio de hoy

- Actualmente D-BUS está muy cerca de la versión 1.0 y está siendo usado de forma exitosa tanto en GNOME como en KDE.
- KDE acaba de anunciar que pasan a usar D-BUS como su principal sistema IPC en el escritorio.
- En GNOME poco a poco se va imponiendo el uso de D-BUS y se están portando partes del escritorio que utilizan bonobo.
- No solo en los sitemas de escritorio tradicionales se usa hoy en día D-BUS, en el Nokia 770 es una de las principales tecnologías junto a otras de la plataforma GNOME.

Conclusiones

- D-BUS es un sistema que en realidad puede parecer que no ofrece nada nuevo. Pero tiene un beneficio extra: la posibilidad de aprender de los sistemas existentes para coger los buenos aspectos y evitar los no tan buenos.
- D-BUS está diseñado con unos objetivos muy concretos y limitados, no pretende ser un estándar genérico como CORBA válido para todas las situaciones imaginables
- La principal ventaja que ofrece D-BUS con respecto a cualquier otro sistema similar es la sencillez. Esta facilidad de uso de D-BUS se debe en gran medida a los bindings.
- D-BUS es una tecnología de freedesktop, lo que facilita el camino hacia la estandarización. GNOME y KDE están usando ya D-BUS

Referencias

- D-BUS Specification: http://dbus.freedesktop.org/doc/dbus-specification.html
- D-BUS Tutorial: http://dbus.freedesktop.org/doc/dbus-tutorial.html
- D-BUS FAQ: http://dbus.freedesktop.org/doc/dbus-faq.html
- D-BUS Source Code: http://dbus.freedesktop.org/doc/api/html/files.html

Preguntas?