eZ
Share your information

Welcome!

PHP on the D-BUS

Dutch PHP Conference - Amsterdam, the Netherlands

Derick Rethans - dr@ez.no - twitter: @derickr

http://derickrethans.nl/talks.php

http://joind.in/580

What is DBUS?

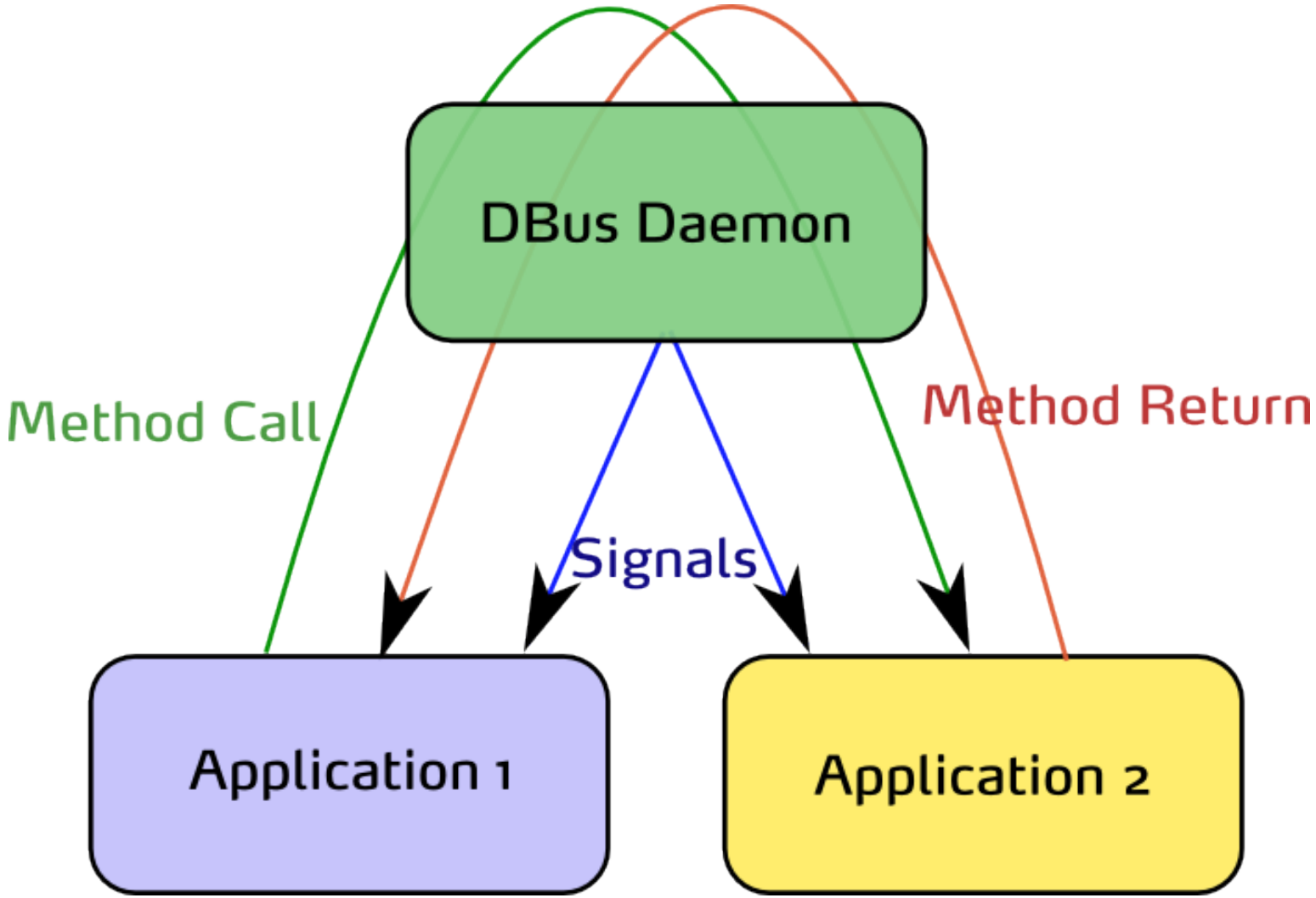
- Part of the freedesktop.org project
- It is meant to replace Corba (GNOME) and DCOP (KDE)
- A message bus system, a simple way for applications to talk to one another
- It consists of a deamon, a library and a protocol
- It is both meant for system level and session level communication

freedesktop.org

What is DBUS used for?

- On the (GNOME) desktop: Avahi, CUPS, GConf, Nautilus, Network Manager, Power Manager, Screen Saver, Volume Manager
- System level elements: HAL
- Gnome applications: Pidgin, Rhythmbox
- Other applications: skype
- But also on the Openmoko platform (freesmartphone.org)

Architecture



Message Types

Method calls

- From one application to another

Method returns

- Upon succesfull method execution

Method error returns

- Upon a failure while executing the method

Signals

- Broadcast from one application to all others

eZ
Share your information

Identifying objects

- Bus: Communication pathways that clients can transport messages over
- Connection: Every connection to the bus has one (or more) unique names (random and well-known) (:1.363 or nl.derickrethans.test).
- Object: A communications end-point that a process exports to offer its services (/nl/derickrethans/test).
- Proxy: A client-level representation of an object on the bus.
- Method: A function on an object; they have input and output parameters and are called through proxies.
- Signal: One way communications from an object to registered clients.
- Members: A collection of methods and signals.
- Interface: A collection of members; each object can implement multiple interfaces (nl.derickrethans.test, org.freedesktop.DBus.Introspectable).

Data-types

Basic types

- byte
- boolean (0 = false, 1 = true)
- integer (int16, uint16, int32, uint32, int64 and uint64)
- double (IEEE 754 double)
- string (UTF-8 string)

Compound types

- array (contains elements of the same type)
- struct (contains one or more complete types)
- dictionary entry (contains a key (basic type) and a value, must be part of an array)
- variant (contains a type and a value)

Signatures

Each method call has a signature for incoming and outgoing parameters

Examples:

- ii: two 32bit signed integers
- tn: a 64bit unsigned integer, and a 16bit signed integer
- ssiib: two strings, two integers and a byte
- ad: an array of doubles (each array has one type following to determine the child's type)
- a{sv}: an array containing dictionary entries with a string key, and a variant value
- a(iiss): an array containing a struct with two integers and two strings
- aai: an array of arrays containing integers

PHP DBUS extension

- DBUS bindings using the low-level C library (libdbus)
- Provides proxy objects
- Implements automatic-marshalling of data
- Provides specific classes for full control
- Support for method invocation, method calls, sending and receiving signals
- (Basic) support for introspection

Talking to a DBUS object from PHP

Activating the screensaver

screensaver.php:

```
<?php
$d = new Dbus;
$n = $d->createProxy(
    "org.gnome.ScreenSaver",
    "/org/gnome/ScreenSaver",
    "org.gnome.ScreenSaver"
);

var_dump($n->GetActive());
$n->SetActive( true );
var_dump($n->GetActive());
sleep(5);
$n->SetActive( false );
?>
```

eZ
Share your information

Data-type mapping
From PHP to DBUS

Basic types

- boolean -> boolean
- integer -> int32
- double -> double
- string -> string
- array -> array (only basic types as values are supported)

Specific type classes

- DbusByte, DbusBool, DbusInt16, DbusUInt16, DbusInt32, DbusUInt32, DbusInt64, DbusUInt64, DbusDouble
- DbusArray(int $type, array $elements [, string $signature])
- DbusDict(int $type, array $elements)
- DbusVariant(mixed $data)
- DbusStruct(string $signature, array $elements)
- DbusSet(array $elements)

Data-type mapping
From DBUS to PHP

Basic types

- boolean -> boolean
- byte, int16, uint16, int32, uint32, int64, uint64 -> integer
- double -> double
- string -> string

Compound types

- array -> DbusArray or DbusDict (depending on whether there is a dict-entry embedded)
- dict-entry -> variable (could be anything)
- variant -> DbusVariant
- struct -> DbusStruct

Talking to a DBUS object from PHP

Using explicit integer types

notify.php:

```
<?php
$d = new Dbus( Dbus::BUS_SESSION );
$n = $d->createProxy(
    "org.freedesktop.Notifications", // connection name
    "/org/freedesktop/Notifications", // object
    "org.freedesktop.Notifications" // interface
);
$id = $n->Notify(
    'Testapp', new DBusUInt32( 0 ), // app_name, replaces_id
    'iceweasel', 'Testing http://ez.no', 'Test Notification', // app_icon, summary, body
    new DBusArray( DBus::STRING, array() ), // actions
    new DBusDict(                           // hints
        DBus::VARIANT,
        array(
            'x' => new DBusVariant( 500 ), // x position on screen
            'y' => new DBusVariant( 500 ), // y position on screen
            'desktop-entry' => new DBusVariant( 'rhythmbox' )
        )
    ),
    1000 // expire timeout in msec
);
echo $id[0], "\n";
?>
```

Introspection

- Most services provide information about their methods' arguments through org.freedesktop.DBus.Introspectable.
- The DBUS extension currently switches between i and u only.

```
<!DOCTYPE node PUBLIC "-//freedesktop//DTD D-BUS Object Introspection 1.0//EN"
"http://www.freedesktop.org/standards/dbus/1.0/introspect.dtd">
<node>
<interface name="org.freedesktop.Notifications">
    <method name="Notify">
    <arg name="app_name" type="s" direction="in"/>
    <arg name="id" type="u" direction="in"/>
    <arg name="icon" type="s" direction="in"/>
    <arg name="summary" type="s" direction="in"/>
    <arg name="body" type="s" direction="in"/>
    <arg name="actions" type="as" direction="in"/>
    <arg name="hints" type="a{sv}" direction="in"/>
    <arg name="timeout" type="i" direction="in"/>
    <arg name="return_id" type="u" direction="out"/>
    </method>
    <method name="GetServerInformation">
    <arg name="return_name" type="s" direction="out"/>
    <arg name="return_vendor" type="s" direction="out"/>
    <arg name="return_version" type="s" direction="out"/>
    <arg name="return_spec_version" type="s" direction="out"/>
    </method>
    <method name="GetCapabilities">
    <arg name="return_caps" type="as" direction="out"/>
    </method>
    <method name="CloseNotification">
    <arg name="id" type="u" direction="in"/>
    </method>
```

Talking to a DBUS object from PHP

Using introspection to select the correct integer type

notify-introspect.php:

```
<?php
$d = new Dbus( Dbus::BUS_SESSION, true );
$n = $d->createProxy(
    "org.freedesktop.Notifications", // connection name
    "/org/freedesktop/Notifications", // object
    "org.freedesktop.Notifications" // interface
);
$id = $n->Notify(
    'Testapp', 0, // app_name, replaces_id
    'iceweasel', 'Testing http://ez.no', 'Test Notification', // app_icon, summary, body
    new DBusArray( DBus::STRING, array() ), // actions
    new DBusDict(                           // hints
        DBus::VARIANT,
        array(
            'x' => new DBusVariant( 500 ), // x position on screen
            'y' => new DBusVariant( 500 ), // y position on screen
            'desktop-entry' => new DBusVariant( 'rhythmbox' )
        )
    ),
    1000 // expire timeout in msec
);
echo $id[0], "\n";
?>
```

eZ
Share your information

Providing DBUS services through PHP

- Request a connection name
- Register a class that provides methods
- Introspection is not yet supported

```
<?php
$d = new Dbus( Dbus::BUS_SESSION, true );
$d->requestName( 'nl.derickrethans.test' );
$d->registerObject(
    '/nl/derickrethans/test', 'nl.derickrethans.test', 'testClass' );

class testClass {
    static function echoOne( $a ) {
        return $a;
    }
    static function echoTwo( $a, $b ) {
        return new DbusSet( $a, $b );
    }
}

do {
    $s = $d->waitLoop( 1000 );
} while ( true );
?>
```

eZ
Share your information

Watching signals

```
<?php
$d = new Dbus;
$d->addWatch( 'org.freedesktop.PowerManagement.Backlight' );
$d->addWatch( 'nl.derickrethans.Interface' );

do
{
    $s = $d->waitLoop( 1000 );
    if ( !$s ) continue;

    if ( $s->matches(
        "org.freedesktop.PowerManagement.Backlight", "BrightnessChanged" )
    {
        $b = $s->getData();
        echo "Brightness: {$b[0]}\n";
    }

    if ( $s->matches( 'nl.derickrethans.Interface', 'TestSignal' ) )
    {
        var_dump( $s->getData() );
    }
}
while ( true );
?>
```

eZ
Share your information

Sending signals

```
<?php
$d = new Dbus( Dbus::BUS_SESSION, true );
$d->requestName( 'nl.derickrethans.test' );
$s = new DbusSignal(
    $d,
    '/nl/derickrethans/SignalObject',
    'nl.derickrethans.Interface',
    'TestSignal'
);
$s->send( "ze data", new DBusArray( Dbus::STRING, array( 'one', 'two' ) ) );
?>
```

Skype

- Implements (on Linux) a DBUS API for communications
- Documented at https://developer.skype.com/Docs/ApiDoc/src (sparsely)

Skype
Objects

- USER object
- PROFILE object
- CALL object
- MESSAGE object
- CHAT object
- CHATMEMBER object
- CHATMESSAGE object
- VOICEMAIL object
- SMS object
- APPLICATION object
- GROUP object
- FILETRANSFER object

Skype
Odd DBUS API

- Everything in and out happens with a weird string protocol
- It does not make use of objects at all
- It doesn't send signals, but you have to provide a callback

eZ
Share your information

Skype in polling mode

```
<?php
$d = new Dbus( Dbus::BUS_SESSION, true );
$n = $d->createProxy( "com.Skype.API", "/com/Skype", "com.Skype.API");
$n->Invoke( "NAME PHP" );
$n->Invoke( "PROTOCOL 7" );
$chatId = $n->Invoke( "CHAT CREATE omsmestad" );
@list( $ignore, $id, $stuff, $stuff2 ) = explode( " ", $chatId );
$n->Invoke( "OPEN CHAT $id" );

while ( true )
{
    $r = $n->Invoke( "GET CHAT $id RECENTCHATMESSAGES" );
    @list( $ignore, $dummy, $dummy, $messageIds ) = explode( ' ', $r, 4 );
    foreach( explode( ", ", $messageIds ) as $messageId )
    {
        $data = $n->Invoke( "GET CHATMESSAGE $messageId FROM_HANDLE" );
        list( $a, $b, $c, $name ) = explode( ' ', $data, 4 );
        $data = $n->Invoke( "GET CHATMESSAGE $messageId BODY" );
        list( $a, $b, $c, $body ) = explode( ' ', $data, 4 );
        echo $name, ": ", $body, "\n";
        $n->Invoke( "SET CHATMESSAGE $messageId SEEN" );
    }
    sleep( 1 );
}
?>
```

eZ
Share your information

Skype with callback

Instead they should have used signals...

```
<?php
$d = new Dbus( Dbus::BUS_SESSION, true );
$n = $d->createProxy( "com.Skype.API", "/com/Skype", "com.Skype.API");
$n->Invoke( "NAME PHP" );
$n->Invoke( "PROTOCOL 7" );
$n->Invoke( "CHAT CREATE omsmestad" );
@list( $ignore, $id, $stuff, $stuff2 ) = explode( " ", $chatId );
$n->Invoke( "OPEN CHAT $id" );

class testClass {
    static function notify($a) {
        global $n;

        @list( $a, $b, $c, $d ) = explode( ' ', $a, 4 );
        if ( $a === "CHATMESSAGE" && ( $d === "RECEIVED" || $d == "SENT" ) )
        {
            $data = $n->Invoke( "GET CHATMESSAGE $b FROM_DISPNAME" );
            @list( $a, $b, $c, $person ) = explode( ' ', $data, 4 );

            $data = $n->Invoke( "GET CHATMESSAGE $b BODY" );
            @list( $a, $b, $c, $body ) = explode( ' ', $data, 4 );
            echo $person, ': ', $body, "\n";
        }
    }
}

$d->registerObject( '/com/Skype/Client', 'com.Skype.API.Client', 'testClass' );

do {
    $s = $d->waitLoop( 1000 );
}
while ( true );
?>
```

Pidgin

Implements things through a proper interface

- It exposes everything through some form of objecty model
- A new method for every functionality
- Uses IDs to identify objects
- Uses signals extensively
- It's not documented very well, but it matches the internal C API

eZ
Share your information

Pidgin example

Listing which buddies are online

```
<?php
$d = new Dbus;
$n = $d->createProxy(
    "im.pidgin.purple.PurpleService",
    "/im/pidgin/purple/PurpleObject",
    "im.pidgin.purple.PurpleInterface"
);

$data = $n->PurpleAccountsGetAllActive();
foreach( $data->getData() as $account )
{
    $buddies = $n->PurpleFindBuddies( $account, '' );
    $protocol = $n->PurpleAccountGetProtocolName( $account );
    echo $protocol, "\n";
    foreach ( $buddies->getData() as $buddyId )
    {
        $online = $n->PurpleBuddyIsOnline( $buddyId );
        $alias = $n->PurpleBuddyGetAlias( $buddyId );
        if ( $online )
        {
            printf( "- %s\n", $alias );
        }
    }
}
?>
```

eZ®
Share your information

What's the real goal?

- Running apps on the OpenMoko
- With PHP-GTK

Questions