DBusKit
Integrating GNUstep Applications with ‘Foreign’ Desktop Environments

Niels Grewe

February 5th, 2011
Distributed Objects

- OpenStep/GNUstep/Cocoa specific IPC system
- Design philosophy: It shouldn’t matter whether an object was created in a different process, or even on a different machine.
- Obtain a proxy for the remote object and use it like a local object:
Distributed Objects

- OpenStep/GNUstep/Cocoa specific IPC system
- Design philosophy: It shouldn’t matter whether an object was created in a different process, or even on a different machine.
- Obtain a proxy for the remote object and use it like a local object:

```cpp
NSPort *sp = [[NSPortNameServer systemDefaultPortNameServer] portForName: @"RemoteService"];
NSConnection *c =
    [NSConnection connectionWithReceivePort: [NSPort port] sendPort: sp];
id remoteObject = [c rootProxy];
[remoteObject doStuffAsUsual];
```
Distributed Objects

- OpenStep/GNUstep/Cocoa specific IPC system
- Design philosophy: It shouldn't matter whether an object was created in a different process, or even on a different machine.
- Obtain a proxy for the remote object and use it like a local object:
  ```objective-c
  NSPort *sp = [[NSPortNameServer systemDefaultPortNameServer] portForName: @"RemoteService"];
  NSConnection *c =
      [NSConnection connectionWithReceivePort: [NSPort port] sendPort: sp];
  id remoteObject = [c rootProxy];
  [remoteObject doStuffAsUsual];
  - Elegant, easy to use, intelligent
Why support D-Bus as well?

- **Wide adoption:**
  - HAL/UDisks/UPower
  - Bluez
  - NetworkManager
  - Avahi
  - GeoClue
  - Gnome
  - KDE

- Allows deeper integration into non-GNUstep desktop environments through standard services, e.g.:
  - org.freedesktop.ScreenSaver
  - org.freedesktop.PowerManagement
  - org.freedesktop.Notifications
D-Bus Concepts

**Bus:** D-Bus runs as a daemon that acts as a name service and as a message broker between applications.

**Service:** Every application on the bus acts as a service that gets one unique name and can request additional names (e.g. ‘org.foo.TextEditor’).

**Object path:** Every service exposes all vended objects in an explicit tree structure.

**Interface:** Methods that can be called on objects are aggregated in interfaces (think Objective-C protocols, but with polymorphism).

**Signal:** Broadcast information is delivered through signals to subscribing applications.
## Comparison: GNUstep DO vs. D-Bus

<table>
<thead>
<tr>
<th>Feature</th>
<th>Distributed Objects</th>
<th>D-Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>type system</td>
<td>native Objective-C type system</td>
<td>custom D-Bus type system (C-like)</td>
</tr>
<tr>
<td>supported programming languages</td>
<td>Objective-C</td>
<td>many languages through bindings</td>
</tr>
<tr>
<td>polymorphism</td>
<td>no special provisions</td>
<td>through overloaded method names in different interfaces</td>
</tr>
<tr>
<td>object-graph generation</td>
<td>implicit</td>
<td>explicit with named objects</td>
</tr>
<tr>
<td>name service</td>
<td>provided by separate nameserver objects</td>
<td>integrated</td>
</tr>
<tr>
<td>delivery of broadcast information</td>
<td>distributed notification system implemented on top of DO</td>
<td>integrated as D-Bus signals</td>
</tr>
</tbody>
</table>


DBusKit: Bringing D-Bus to GNUstep

- Using the libdbus low-level API adds needless complexity and repetition.
DBusKit: Bringing D-Bus to GNUstep

- Using the libdbus low-level API adds needless complexity and repetition.
- Example 1: Using libdbus, etoile_system needs 100 loc of boiler-plate code just to call the suspend/shutdown/reboot methods of HAL.
Using the libdbus low-level API adds needless complexity and repetition.

Example 1: Using libdbus, etoile_system needs 100 loc of boiler-plate code just to call the suspend/shutdown/reboot methods of HAL.

Example 2: Zeroconf/Bonjour support in gnustep-base is provided by Avahi. The Avahi C-API is a superficial wrapper around the D-Bus APIs. Using the wrapper layer accounts for roughly 17% (=430 loc) of all Avahi-related code in gnustep-base.
DBusKit: Bringing D-Bus to GNUstep

- Using the libdbus low-level API adds needless complexity and repetition.
- Example 1: Using libdbus, etoile_system needs 100 loc of boiler-plate code just to call the suspend/shutdown/reboot methods of HAL.
- Example 2: Zeroconf/Bonjour support in gnustep-base is provided by Avahi. The Avahi C-API is a superficial wrapper around the D-Bus APIs. Using the wrapper layer accounts for roughly 17% (=430 loc) of all Avahi-related code in gnustep-base.
- GNUstep needs proper bindings for D-Bus:
DBusKit: Bringing D-Bus to GNUstep

- Using the libdbus low-level API adds needless complexity and repetition.
- Example 1: Using libdbus, etoile_system needs 100 loc of boiler-plate code just to call the suspend/shutdown/reboot methods of HAL.
- Example 2: Zeroconf/Bonjour support in gnustep-base is provided by Avahi. The Avahi C-API is a superficial wrapper around the D-Bus APIs. Using the wrapper layer accounts for roughly 17% (=430 loc) of all Avahi-related code in gnustep-base.
- GNUstep needs proper bindings for D-Bus:

  DBusKit
DBusKit Quick Facts

- LGPL licensed.
- Roughly 8k loc, at the moment.
- Complete implementation of outgoing D-Bus support (you can use D-Bus objects from Objective-C code without limitation).
- Follows Objective-C conventions (like Distributed Objects) as closely as possible.
- No release yet. (coming soon!)
DBusKit Architecture

NSRunLoop-Integration Layer

▶ Handles interaction with the D-Bus daemons using libdbus primitives
▶ Presently being rewritten to support robust multithreaded operation

D-Bus↔Objective-C Translation Layer

▶ Uses D-Bus introspection data to map D-Bus entities to their Objective-C equivalents:
  ▶ D-Bus interface → Objective-C protocol
  ▶ D-Bus object-path nodes → DKProxy (NSProxy subclass for use with D-Bus)
  ▶ D-Bus properties → Accessor/mutator methods
  ▶ D-Bus signal → NSNotification
  ▶ NSInvocation → D-Bus Method call
  ▶ D-Bus Method reply → NSInvocation

DO Convenience Layer

▶ Provides NSConnection methods and the DKPort class to provide a familiar abstraction for Objective-C programmers.
Flexible method generation

D-Bus XML introspection data:

```xml
<method name="NameHasOwner">
  <arg direction="in" type="s"/>
  <arg direction="out" type="b"/>
</method>
```

Is turned into:

- (NSNumber*)NameHasOwner: (NSString*)argument1;
- (BOOL)NameHasOwner: (char*)argument1;
- (BOOL)NameHasOwner: (NSString*)argument1;
- (NSNumber*)NameHasOwner: (char*)argument1;

Note: Only for free software runtimes, Apple's runtime lacks typed selectors.
Flexible method generation

D-Bus XML introspection data:

```xml
<method name="NameHasOwner">
  <arg direction="in" type="s"/>
  <arg direction="out" type="b"/>
</method>
```

Is turned into:

- `(NSNumber*)NameHasOwner: (NSString*)argument1;
- `(BOOL)NameHasOwner: (char*)argument1;
- `(BOOL)NameHasOwner: (NSString*)argument1;
- `(NSNumber*)NameHasOwner: (char*)argument1;`
Flexible method generation

D-Bus XML introspection data:

```
<method name="NameHasOwner">
  <arg direction="in" type="s"/>
  <arg direction="out" type="b"/>
</method>
```

Is turned into:

- `(NSNumber*)NameHasOwner: (NSString*)argument1;`
- `(BOOL)NameHasOwner: (char*)argument1;`

Note: Only for free software runtimes, Apple's runtime lacks typed selectors.
Flexible method generation

D-Bus XML introspection data:

<method name="NameHasOwner">
  <arg direction="in" type="s"/>
  <arg direction="out" type="b"/>
</method>

Is turned into:

- (NSNumber*)NameHasOwner: (NSString*)argument1;
- (BOOL)NameHasOwner: (char*)argument1;
- (BOOL)NameHasOwner: (NSString*)argument1;

Note: Only for free software runtimes, Apple’s runtime lacks typed selectors.
Flexible method generation

D-Bus XML introspection data:

```xml
<method name="NameHasOwner">
  <arg direction="in" type="s"/>
  <arg direction="out" type="b"/>
</method>
```

Is turned into:

- `(NSNumber*)NameHasOwner: (NSString*)argument1;
- `(BOOL)NameHasOwner: (char*)argument1;
- `(BOOL)NameHasOwner: (NSString*)argument1;
- `(NSNumber*)NameHasOwner: (char*)argument1;

Note: Only for free software runtimes, Apple's runtime lacks typed selectors.
Flexible method generation

D-Bus XML introspection data:

```xml
<method name="NameHasOwner">
  <arg direction="in" type="s"/>
  <arg direction="out" type="b"/>
</method>
```

Is turned into:

- `(NSNumber*)NameHasOwner: (NSString*)argument1;
- `(BOOL)NameHasOwner: (char*)argument1;
- `(BOOL)NameHasOwner: (NSString*)argument1;
- `(NSNumber*)NameHasOwner: (char*)argument1;

Note: Only for free software runtimes, Apple’s runtime lacks typed selectors.
Using DBusKit

Make Method Calls

DKPort *sp =
    [[DKPort alloc] initWithRemote: @"org.freedesktop.DBus"
onBus: DKDBusSessionBus];
DKPort *rp = [DKPort sessionBusPort];
NSConnection *c = [NSConnection connectionWithReceivePort: rp
    sendPort: sp];
id remote = [c proxyAtPath: @"org/freedesktop/DBus"]; NSArey *peers = [remote ListNames];
Using DBusKit

Make Method Calls

DKPort *sp =
    [[DKPort alloc] initWithRemote: @"org.freedesktop.DBus"
onBus: DKDBusSessionBus];
DKPort *rp = [DKPort sessionBusPort];
NSConnection *c = [NSConnection connectionWithReceivePort: rp
    sendPort: sp];
id remote = [c proxyAtPath: @"org/freedesktop/DBus"];
NSArray *peers = [remote ListNames];

Receive Notifications

id myObject = [MYObject new];
DKNotificationCenter *center =
    [DKNotificationCenter sessionBusCenter];
[center addObserver: myObject
    selector: @selector(didReceiveNotification:)
    signal: @"NameAquired"
    interface: "org.freedesktop.DBus"
    sender: nil
    destination: nil];
Demo 1

Apertium Service
(DBusKit related code: 10 out of 600 loc)
Demo 2

Desktop Notifications in SimpleAgenda
(DBusKit related code: 82 out of 7615 loc)
Future Plans

- Release
- Asynchronous method calls
- Vending objects to D-Bus
- D-Bus menus
Acknowledgements

Funding:
Google

Mentorship:
Fred Kiefer

Bug Reports:
Philippe Roussell
Acknowledgements

**Funding:**
Google

**Mentorship:**
Fred Kiefer

**Bug Reports:**
Philippe Roussell

**Patient Listening:**
You!