Desktop on the Linux – are we doing it right?

Wolfgang Draxinger

2010-12-14

For a long time Linux and in a broader sense all *nices were said to lack proper desktop environments. The last years showed different. Today a Linux or Unix user can choose from a number of feature rich desktop environments, which don't have to stand behind commercial paragons. The development of the major FOSS DEs, Gnome and KDE and in their wake the freedesktop.org standards were mostly focused on getting hold in the userbased of those closed source desktop systems and unfortunatel many fallacies found in their commercial counterparts sneaked into the FOSS DE's development process.

This text intends to highlight those unlucky spots in the hope to avoid making the same errors again. It is written from the view of a university's faculty student computer systems administrator with a 3.5k userbase and about 100 installed machines. Since this is a personal view it's highly opinionated, but all criticism is based on disturbing experiences made with the most recent versions of modern Linux distributions and their default desktops environments.

Complexity has nothing to do with intelligence, simplicity does. – Larry Bossidy

1 The Big Picture

A modern desktop consists of tremendous amount of software. System and session communicators, on demand service launchers, session seat and console managers, authenticator services, login managers, multimedia libraries, notification services, hotplug device identification, removeable media volume access...

While many of those tasks mentioned have been traditionally carried out by a grown and evolved set of utilities and daemons most of them are being replaced by the efforts of desktop developers. Sometimes for good reason, but more often than not because of a *not invented here* state of mind.

In the typical contemporary Linux system one will find the following pieces alongside what most people may or may not know:

- **system dbus**
- **consolekit**
- **policykit**
- udev
- **hal**

- **gconf**
- networkmanager
- powermanager

Looking at what's currently in the pipeline those may be sided with:

- realtimekit
- systemd

Additionally each opened session will start a bunch of services themself:

- **session dbus**
- semantic desktop
- multimedia
- notifications

Each of those pieces came into life within the last 4 to 6 years. While each of the mentioned parts has it's caveats, it are those written in bold letters, which are, in my humble opinion, flawed.

2 D-Bus

D-Bus came into life as a generalized replacement of DE's propriatary IPC and notification facilities. Namely *dcop* of the KDE and *Bonobo*[1] of Gnome.

Within a session it is for example used to message the session manager of the user's request to terminate and logout the current session. Another use is, to pass filesystem and network operations to a service daemon, executing the requested tasks on the behalf of a file manager.

[1]It shall be noted, that Bonobo offered significatly more, than just passing messages. However it was mostly used for just that.

D-Bus soon grew out from intra session message passing to an exchange between system services and a user's session. Mounting a removeable media or managing network configuration are among the high profile applications.

One of the main claims of initial dbus development was to provide a unified API through which programs could exchange data with arbitrary peers. So from a far point of view D-Bus, or a facility like D-Bus seems like a good idea. Unfortunately the implementation is flawed.

2.1 Questionable Issues with D-Bus

2.1.1 Lack of built in encryption and salting

Although it's design is transport neutral, so far D-Bus communication mostly happens over Unix domain sockets only – POSIX permissions and ACLs are used for access control. There is a TCP transport module available, but it lacks proper end-to-end encryption. However D-Bus is also used for *authentication* and *authorization* applications using a SASL scheme. The lack of proper encryption support despite its use in security critical tasks single handedly prohibits its use over networks.

In its current design and implementation D-Bus is susceptible to replay attacks, unless the peering programs account for that. Any claim of D-Bus being fit for networked applications is this either the product of lack of information or ignorance. ConsoleKit and PolicyKit, both security critical services, are depending on D-Bus nevertheless.

2.1.2 Narcistic Namespacing

Services within D-Bus are identified and accessed by reversed domain names, kind of like they're used for Java packages. One common namespace e.g. is org.freedesktop, org.kde and org.gnome. Essentially every program that's to use D-Bus must have such a domain. While there's no direct relation between Internet domain names and D-Bus the recommended practice is, to use one's application domain name for identification. The reasoning behind this is, that this avoids namespace clashes.

In fact this approach is best described as *narcistic naming*: Instead of grouping programs and services by their function and the environment they operate in, it is only the program's name, or that of it's creator identifying it. The result is, at best, confusing. Even worse it causes major disruptions, if a program's name is changed, either voluntarily or by force. Since the namespace is merely sorted by name, not by function, the interfaces for each service are arbitrary and strongly depend on the actual service provider.

A far better naming scheme would have sorted services by their task and environment. Within each of these a specific interface which participants followed would allow to freely exchange equivalent service providers without explicit dependencies between service provider and consumer.

A very good example of a function sorted namespace is the *SysFS* of the Linux kernel. SysFS is easily readable and traversable not only in naming but also by sense by both humans and machines.

2.1.3 The Hammer of Freedesktop

"To someone with only a hammer for a tool every problem looks like a nail", this proverb aptly described the development patterns of almost every Freedesktop project. Almost every Freedesktop standard depends in one way or another on D-Bus no matter if even a separation in parts makes sense for the whole project. Examples of such cases are given further down.

2.1.4 Scalability problems

Although D-Bus is in the third generation of IPC mechanisms it hit scalability problems. A proposed solution is to extend the Linux kernel for supporting a new socket type[1].

2.2 Proposal for an alternative

In preparation of the talk I learned about a much nicher approach on the problem, which would also solve the ongoing network transparency issues: *IPv6 local multicast*. Due to the builtin IPSec support and the versatile routing capabilites of IPv6 this would singlehandedly tackle both the scalability and transport security problems.

3 ConsoleKit

ConsoleKit is probably the most diffuse part of a Freedesktop system. It seems that even its main developers don't have a clear idea for its purpose. At the moment of writing this article, the latest version of the ConsoleKit main documentation the section about it purposes is:
Defining the Problem
To be written.
In its current implementation ConsoleKit tracks user sessions, being aware of multi-seat configurations, does bookkeeping and permission adjustments on system resources available to the user.

All in all ConsoleKit mostly behaves like a utmp/wtmp on D-Bus, with the additional capability of connecting to the DE's session manager and being aware of the X11 display.

3.1 Problematic method of user access control

Among other thigns ConsoleKit is also responsible for granting and denying users access to system services, based on which *seat* they're logged on and if their session is active.

A good example would be access to the sound devices:

On a traditional *nix system, each class of devices has permissions 0660 and is owned by a device specific group. Users granted access to these devices are either added to the devices groups explicitly, or on the fly upon login depending on their main group and the terminal of their login. In the later case only processes originating from a granted terminal login process group can access the device.

Contrary to this, ConsoleKit manages permissions on the devices itself ACL for the user in question. Depending on the configuration ConsoleKit may change the permissions on devices at any time. This for example happens when switching the VT.

There is problem with this: Since filesystem level permissions and ACLs only apply upon opening a filedescriptor, once opened a device is in the process hold until the FD is closed. In a ConsoleKit based system programs are thus required to release devices on request, any may very well be forcefully terminated after some timeout. In the case of a sound device getting your sound application terminated, just because one switched to another VT may disrupt a recording session, but it's annoying at least. If however the device in question is a serial interface such behaviour may cause serious damage, for example if a firmware update process gets interrupted in this way.

3.2 Multiseating very uncommon at the moment

So far multiseated systems are very rare. This by itself is no argument against ConsoleKit or the need for multiseating. There is just a lack of experience on the needs within a multiseated environment to make educated choices about designing a seat management system like ConsoleKit.

4 PolicyKit

PolicyKit is a system to enable system level processes to request authorizing privileged tasks by the user. Like any other IPC based software under the hood of Freedesktop it uses D-Bus for data exchange. In its purpose and user experience PolicyKit oftenly is compared to *sudo*.

4.1 PolicyKit authorizes, sudo escalates

Whichever process requests authorization by means of PolicyKit, the process performing the actual task or spawning such process is running with the required privileges already before asking for authorization. It is to be expected that the majority of to-be-authorized tasks are initiated by unprivileged users by means of a IPC messaging, probably over D-Bus. If however the executing process erroneously interprets the commands given over D-Bus this may result in privileged execution even before the authorization is given.

Compared to this, sudo checks the authority of the user to perform an action before the target process is even started. The target process may still be exploitable by malicious control input, but unlike with PolicyKit a targetable process is only executed on demand and terminates early.

While this is not a flaw in the design on PolicyKit itself, the whole system is based on the assumption that the IPC mechanism used is secure and can't be exploited. As already pointed out the system in question, D-Bus should not be considered secure at the moment.

5 HAL

There's little to say about HAL, except that it's dying. HAL has been marked as deprecated by its original creators. Unfortunately it's still being used widely and even new software is written against it.

The original meaning of HAL is *Hardware Abstraction Layer*. Being located in userspace and not residing between the user and the hardware like the kernel does, there is no abstraction whatsoever. A better backronym of HAL thus would be *Hardware Annotation Library*, which is what actually is. HAL is a huge monster of countless XML files, obscure rules and methods aimed at the task to give programs information what a certain piece of hardware does. The irony is, that HAL, while shipping with a large dataset, gets most of its information from SysFS. SysFS OTOH provides exactly the abstraction to the system's hardware one would expect from a HAL.

This is just a reminder, that HAL is no longer maintained properly and if you're still relying on it in your programs you should phase it out. Whatever you intend, doing it through SysFS/UDev is much more viable.

6 GConf

GConf and its sibling Elektra both started with the idea of providing applications a common configuration API and storage. Essentially it is the concept of the Windows Registry ported to *nices. As such is suffers they suffer from the same flaws.

In case of GConf it gets worse: gconf has a huge set of dependencies, depending on configuration and used modules even down to having a full session running. This is one of the main reasons for the all new ¿GDM-2.21 starting a full Gnome session for a simple login window: The whole configuration has beed placed in GConf, so to function properly a Gnome session must be started before the login greeter can be shown.

In summary the problems of a system like GConf have been discussed elaborately, any criticism on, say the Windows registry apply on GConf as well. The leaner and preferrable solution consists of a leightweight configuration file access library and some well known directories into which to place configuration files. Luckily both of such exist and are used.

7 Lock-In effect

One aspect of the Freedesktop infrastructio one may not forget is its lock-in effect: In a Freedesktop environment many building blocks are built upon. ConsoleKit depends on D-Bus, PolicyKit depends on ConsoleKit, the desktop environment depends on PolicyKit and so on. Once a certain level of dependency is established, it's next to impossible to replace or remove certain parts without breaking the whole sys-

tem.

While a certain degree of dependency is acceptable as long they are limited to a confined environment, certain aspects of desktop environment development have started to leak into base operating system design. For example at the moment there are several systems in place to replace the traditional SysV init system. Upstart, einit and systemd are just a few to name. On a similar level are the efforts of creating a special D-Bus socket type in the Linux kernel.

It can be observed that current desktop environments development even tangle with system level service management. It may very well be, that future DEs may depend on a certain init system to function properly.

If this happens as an end user one then ultimately only has the choice between a stringent set of system tools and a desktop environment, or no desktop environment but free choice of system tools

I think it's time to reflect on the way we've come and the decisions we made before we venture further in the unknown of FOSS DE development, for I see that the train may derail.

[1] http://alban.apinc.org/blog/2010/09/15/d-bus-in-the-kernel-faster/