The X11 clipboard

An overview of it's problems and a proposed solution

Philip Van Hoof - http://www.pvanhoof.be

GNOME Foundation member (pvanhoof@gnome.org)

Software developer at X-Tend (vanhoof@x-tend.be)

http://www.x-tend.be - http://www.cronos.be

Table of contents

About

The clipboard has been a subject of many discussions. Often people have had troubles using the clipboard, which aren't problems of the desktop applications. They are inherited from the problematic design of the X11 clipboard protocol.

Other systems, like Microsoft Windows, have faced similar problems. They deprecated their original clipboard infrastructure and now suggest the usage of a clipboard that transfers its data over OLE/DCOM.

I'll quote Mike Hearn from a *mailinglist-discussion of September 2003 [7]* about the Microsoft Windows clipboard:

```
    * Windows has two clipboards, the legacy clipboard which operates as you might
      expect given the discussions here and is stored, well, who knows where, and an
      OLE/DCOM based clipboard. The two interoperate to some extent.

    * Often, both are used at once. For instance, if you select some cells in Excel 97
      and copy them, you have a choice of formats to paste them into Word with,
      including Formatted Text (RTF), and an embedded Excel document. The RTF comes from
      the standard clipboard, and obviously embedded Excel is done via OLE.

    * Many formats are only available via the OLE clipboard while the originating
      application is running. These formats are lost when the app closes. For instance
      while Excel is running, you can paste as a bitmap. If you close it, you can only
      paste as text.

    * Data transfers via OLE are seemingly done by marshaling an an interface specified
      by OLE onto the clipboard as another format type, which is then demarshaled into
      the context of the application being pasted into. DCOM deals with the RPC. These
      formats disappear (removed either explicitly by the app, or by the framework, I
      don't know which) when the source app closes. The interface used here supports
      other forms of content negotiation, as well as the ability to receive updates when
      it changes.

    BTW, the old style clipboard, where the copied data is buffered by the OS, seems to
    be official deprecated:

    Follow these guidelines in using the Clipboard:

            * Use the OLE Clipboard mechanism now to enable new capabilities in the
              future. While the standard Clipboard API will be maintained, the OLE
              mechanism is the future of data transfer.

            * Use the OLE Clipboard mechanism if you are writing an OLE application or
              you want any of the OLE features, such as drag and drop.

            * Use the OLE Clipboard mechanism if you are providing OLE formats.
```

The problems with the clipboard on Microsoft Windows aren't fully solved. Nevertheless that doesn't mean we should keep our clipboard implementation broken, too. It's wrong to think that we shouldn't fix our problems because others have problems too.

Some projects have been created to address some of the problems. Examples are *Klipper* by Carsten Pfeiffer, *Clipboard Manager* and *netclip* by Philip Van Hoof, *Bonobo Clipboard* by Gergõ Érdi and *Clipboard daemon* by Hongli Lai. The fact that desktop application developers eventually decided to implement these problem-solvers, clearly illustrates problems with the current clipboard mechanisms.

I can't imagine a successful windowing system that needs a set of incompatible applications to handle its *drag and drop*. Yet on the X11 platform the clipboard and *drag and drop* concepts are using the same mechanisms. And yet does every popular windowing environment implement such a set of tools.

None of the applications mentioned above interoperate with each other in a correct way. Clearly, there's no standard whatsoever for the fixes being implemented by these tools and windowing environments. With the exception of the *clipboard manager* atom. The problem has been pushed away. It wasn't important because it looked impossible to fix. Today, in contrary, it is becoming a very important problem.

None of the tools mentioned above solve all the problems. In fact, they needed to introduce new problems. Problems like huge memory consumption. Just to solve very basic issues like keeping the clipboard data persist when the clipboard owner decides to shut itself down.

Most of the times the reason for introducing such new problems is trying to achieve *total backwards compatibility*.

In order to get all the problems solved, all the tools mentioned above should become combined. All of them are implemented using incompatible libraries and targeted towards specific windowing environments. It's unlikely this will happen.

The difficulties and problems with the X11 clipboard

In the past there have been several problems with the X11 clipboard. These have been repeated in numerous discussions on various mailinglists like the xdg-mailinglist on freedesktop.org. Some have been separately solved by the tools mentioned above.

Agreed, they've probably been repeated so many times they've become boring. Yet I'm going to list them one more time. It's not because talking about the problems became boring, the problems have disappeared. All of them are still fresh and available on your local X11 implementation. The X.org X11 implementation still has all of the clipboard-related problems which are listed here.

At a first glimpse the problems seem easily solvable. Yet I can, using both technical and political explanation and reasoning, assure you they aren't. For example, don't forget the backwards compatibility. It's very important!

Console applications

It's impossible to copy-paste data between graphical X11 and console applications.

The copy-pasting happens by emulating a terminal (like xterm). This isn't a real solution but rather a hack. Imagine your console editor asking the clipboard information in the format it wishes to receive. Try pasting HTML tags from the document you're viewing on your browser into your HTML console editor (without viewing the HTML source). It's possible on X11 between HTML editing applications. Why can't console applications join the fun?

At this moment it only supports copy-pasting the format that's supported by your terminal software (like xterm). The terminal software acts as a proxy between the clipboard of X11 and the console application. In fact it converts it to keyboard keystrokes,which is slow for larger data-chunks and many editors get line wrapping completely wrong. It's needlessly difficult for such a console application to handle copy-pasting, through a terminal, as keystrokes to the console application correctly.

There's also no standard whatsoever for copy-pasting between two console desktop applications (without an X11 server running). It's not because most application developers don't support clipboard copy-pasting between such a console application, that people don't want it. In fact they do (read the xdg mailinglist). And in fact it's easy to support.

The most important reason why console applications aren't supporting copy-pasting is simply because there's no standard or because there are ad hoc solutions like converting the data to keystrokes by a graphical terminal emulator.

When the session dies the clipboard data is lost

There's no recovery of the clipboard-data after reentering your session. Typical users don't really care much about the security implications of storing the clipboard between sessions. It isn't an important use-case for normal graphical desktop-usage. But it becomes an important one if the clipboard is to be shared across both the graphical desktop applications and the console desktop applications.

When the clipboard owner dies the clipboard data is lost

There's no recovery of the clipboard after an application, that's owning the clipboard, shuts down. Try this test-case:

Launch Mozilla Composer. Launch Gnumeric. Type 1,2,3,4 in Gnumeric. Copy-paste it to Mozilla Composer. That works. Now close Gnumeric and try pasting again. Guess what: no more clipboard!

It's very difficult to implement this, even with the new *xfixes extensions [3]*, because applications can *generate* the requested format. This implies that many applications don't have to keep the data itself buffered during the time they posses ownership of the clipboard.

They can implement a script or algorithm to convert their original format to the many formats they support. Trying to make the clipboard data of a dying clipboard owner application persist would involve large redundant data transfers and memory storage. Each available format needs to be caught and stored. Whereas the dying application itself probably had an algorithm to generate all the available formats on-the-fly. Many applications, like office suites and browsers, support lots of such formats. Lots of formats equals lots of data to recover and to store in an extremely inefficient way.

Clipboard sharing and network transparency

It's nearly impossible to make the clipboard shared across different desktop computers. In fact it is possible, but such an implementation would be needlessly difficult and complex.

The same can be said of support for virtualization (Qemu, Xen, VMWare). Sharing the clipboard between a virtual machine and the desktop itself is painfully difficult to implement correctly (in case X11 is running on the host operating system). Virtual machines are becoming important aspects of desktop computing.

Local clipboard-data transfers when xserver is remote

The transferring of the actual data from the owner to the requester is often to slow for modern uses. Especially when the xserver isn't running on the same machine as the applications are.

Many users want to copy-paste spreadsheet and database data from one application to another. Often these data transfers become quite large in size. You don't want to needlessly send that data over a slow communication link, like TCP/IP. Often applications will run on the same machine, sharing the same CPU and memory. In which case they can transfer their larger data to each other by using a simple Interprocess Communication System that isn't going over a network interface.

The current implementation will always transfer the clipboard data via the xserver. The mechanism is careless about whether or not the xserver is running remote or local. Many modern uses won't work this way. For a normal user it looks like both have crashed, given the fact that both applications (owner and requester) become highly unresponsive during the process.

Formatstandards

There's few agreement on the different formats that can (or should) be supported. Popular formats are the “text/html” and a UTF-8 plaintext format.

Today the situation is improving. Once, a popular E-mail client and a popular browser didn't agree upon the Unicode format of the “text/html” format. Simply because the format wasn't specified. Once, a popular browser only converted its first four kilobytes of clipboard data when a non-supported application requested it.

Luckily these issues have been resolved. But they also partly show the problematic design issues of the X11 clipboard.

Today still many applications implement their clipboard handling incorrect. Many won't support a sharable clipboard format. Many are inconsistent with other applications: Gnumeric and OpenOffice.org's Calc or spreadsheet application will deliver two different HTML tables when pasting them in an HTML editor like Mozilla Composer. There should have been a standard for copyable table-data. Today it's incompatible for many applications, hard to implement and parse for others.

Confusingmultipleclipboards

From *a document on freedesktop.org [2]*:

```
The clipboards that exist on X11 today:
      1. PRIMARY for mouse selection, middle mouse button paste, and explicit
         cut/copy/paste menu items (Qt 2, GNU Emacs 20);
      2. CLIPBOARD for the Windows-style cut/copy/paste menu items;
      3. No one ever does anything interesting with SECONDARY.

The correct behavior can be summarized as follows: CLIPBOARD works just like the
clipboard on Mac or Windows; it only changes on explicit cut/copy. PRIMARY is an
"easter egg" for expert users, regular users can just ignore it; it's normally
pastable only via middle-mouse-click.
The ICCCM [ 4 ] defines these as follows:

"The selection named by the atom PRIMARY is used for all commands  that take only
a single argument and is the principal means of communication between clients that
use  the  selection mechanism.
The selection named by the atom SECONDARY is used:

   1. As   the  second  argument  to  commands  taking  two  arguments  (for  example,
      "exchange  primary  and  secondary  selections");
   2. As  a  means  of  obtaining data when there is a primary selection and the
      user does not want to disturb it.
The selection named by the atom CLIPBOARD is  used  to  hold data  that  is  being
transferred between clients, that is,  data that usually is being cut and then
pasted or copied and then  pasted."
```

The differences between the two (in fact, three) available X11 selections/clipboards are very confusing for normal users. To let a normal user understand these clipboards, the user should first read the ICCCM documents. A graphical environment has the purpose of being intuitive and easy to use.
Try explaining a user how come he copied something using application x and now suddenly the *paste* action of his favorite X11 editor isn't pasting the data. It won't work by showing the user some ICCCM documentation! The user will refuse to comprehend or even read it. And the user is right! A user shouldn't have to read such documentation.

Yet it's a very bad idea to build a clipboard manager or tool that will keep them synchronized.

Such a tool would overwrite the clipboard during typical actions being taken by a user of a graphical desktop. Imagine selecting an URL in your browser. Synchronizing the PRIMARY with CLIPBOARD would now overwrite CLIPBOARD. Whereas the user probably selected that URL to overwrite it with what he has prepared in his clipboard. But that data has just been overwritten!

The concept of the PRIMARY clipboard is broken and SECONDARY isn't used. The only clipboard that really works and does what it's supposed to do, is the CLIPBOARD clipboard.

The same *document on freedesktop.org [2]* also explains why the PRIMARY selection is broken:

```
1. It's inconsistent with Mac/Windows;
2. It's confusingly. Selecting anything overwrites the clipboard;
3. It's not efficient with a tool such as xclipboard;
4. You should be able to select text, then paste the clipboard over it, but that
   doesn't work if the selection and clipboard are the same;
5. The copy menu item is useless and does nothing, which is confusing;
6. If you think of PRIMARY as the current selection, cut doesn't make any sense
   since  the  selection  simultaneously  disappears  and  becomes  the  current
   selection.
```

Backwards compatibility: an important issue

It's hard to solve any of these problems because of one big issue: many application developers like the fact that X11 has a good record on backwards compatibility. Yet are many of the clipboard problems related to both incorrect implementations by many desktop applications *and* historical decisions about the X11 clipboard protocols and mechanism.

In fact, the mechanisms itself are the larger part of the problem. It's easy to say: *It's the damn application developers*. Then please also explain how come other platforms don't have that much problems. Agreed, they have similar problems, but less.

Also note that the solutions built by other desktop platform suppliers don't need to be the limit of quality. We can easily outperform them. And in fact we should. They might have, however, interesting solutions to study. We can and should learn from their and our mistakes.

Proposed solutions for the listed problems

Many ideas on mailinglists, blogs, during mindsharing and during discussions have been formed. Some of them are not interesting. Others ignore or focus to much on the backwards compatibility issue.

A proposed solution shouldn't avoid solving the problems. In fact the target of the solution *is* solving the problems! If only backwards compatibility is important, the question, whether or not a solution is desirable, should be asked.

The proposals below assume the problems need a solution while giving backwards compatibility a high priority. The proposals below are all, in essence, backwards compatible.

Supporting copy-pasting to and from console applications

An IPC system, **like** D-BUS, can be used to transfer clipboard data from one application to another. This doesn't mean that all the application developers should learn about how to use an IPC system like D-BUS. They can easily talk with the clipboard system by using a library like *libclipboard.so*.

In case an application is interested in the current clipboard, the old clipboard-mechanism could be used to let other applications know about the fact that the clipboard owner is *libclipboard.so* -aware. An IPC like DBUS, however, has other possibilities for this (it's optional).

Clipboard persistence between sessions

A plugin for the proposed clipboard manager (read more about this below) can easily be built. Such a plugin can implement a feature like *persisting clipboard history data to the filesystem*. The plugin can then easily recover from that data. A plugin per windowing environment is also a possibility (Support for GNOME sessions differ from KDE sessions in their implementation). An interchangeable clipboard format might also reduce the amount of data to persist. Read more about the proposal for an interchangeable clipboard format below.

Clipboard recovery when the owner dies

When using an IPC like D-BUS, the transfer of all formats will (often) be much faster.

As also proposed in issues one and six: an interchangeable format could be developed and used for this purpose if supported by the desktop application. Just as for clipboard persistence between sessions, can this be built as a plugin for the clipboard manager.

There can be a different or extra approach to let the application developer implement all clipboard functionality and generation of the clipboard in a *virtual machine* or a *different process* or *however* you want to call this. Of course should this be part of *libclipboard.so*. This way that (extra) process can be kept alive until its clipboard ownership is taken away. This isn't an easy solution. But then again, this isn't an easily solvable problem neither.

It implies that an IPC between that virtual machine and core-routines of *libclipboard.so* should exist (to let the desktop application talk with it). It also implies that the virtual machine should easily be linkable with pieces of code (algorithms) of the desktop application developer (plugins). For example document format converting algorithms or *xslt transformations [5]*.

Clipboard sharing and network transparency

The clipboard manager can make it possible for virtual machine software to interact more directly with clipboards by allowing them to plugin their own custom plugins in a clipboard manager.

For network sharing, applications that will seamlessly integrate the clipboard of multiple desktop computers can be more easily developed.

This fourth issue isn't representing commonly requested features. It tries to illustrate that the usage of a clipboard manager (with plugins) can make the clipboard more flexible and more fit for future usage.

Speeding up local clipboard-data transfers when xserver is remote

The usage of an IPC can improve the performance, mainly because the transfer of data doesn't have to happen through the xserver. As we all know, can the xserver be running at a different network location. For example at the graphical desktop terminal hardware of the user.

The usage of a fast Interprocess Communication System can improve the performance of clipboard transfers for applications that are running on the same machine. They can communicate locally. For applications that are running on different hosts while sharing the same xserver (less common but possible situation), the old X11 clipboard is suitable and still available.

Making “TARGET” format standards

There should be an organization suggesting desktop application developers which formats to support. Perhaps an interchangeable clipboard format (or a few) could be developed (based on existing standards like *XML [6]* and *Xslt [5]*). This can also improve the situations described in the third and second issues. Since then only one such interchangeable format needs to be persisted.

Removingconfusingextraclipboards

Deprecating the PRIMARY and SECONDARY clipboards will make it more easy for average users of a graphical desktop. For normal users only one clipboard concept is really important: the CLIPBOARD clipboard.

The name is shared by both the *libclipboard.so* and the legacy world. Yet they are two separated clipboards, connected with each other by means of a hybrid application: the clipboard manager. They are connected with each other to make it possible to let older X11 applications share clipboards with newer *libclipboard.so* applications. Read more about backwards compatibility in the next section.

The old X11 clipboard mechanism, however, could be used to make applications aware of the support for a *libclipboard.so* communication with the current clipboard owner. This way the clipboard manager can easily distinguish between clipboards coming from an old X11 application, and clipboards coming from *libclipboard.so*-aware applications that are trying to support both worlds. Otherwise the clipboard manager would steal that ownership and bridge it with the X11-style clipboard of the current clipboard owner once another application requests it.

Backwardscompatibility

Backwards compatibility deserves its own section. That's because without it, any effort is worthless. It just wouldn't be deployed: not by the distributors nor by the popular windowing environments. Therefor it wouldn't be used by the users.

The clipboard manager can use the new *xfixes extensions [3]* to implement legacy support for the older clipboard protocol. It can easily at the same time talk to the new protocol (so a hybrid application). This way one clipboard manager can copy clipboards of older X11 applications to the “*libclipboard.so*”-world. And can the daemon become clipboard owner playing a bridge between the *libclipboard.so* desktop applications and the older X11 applications:

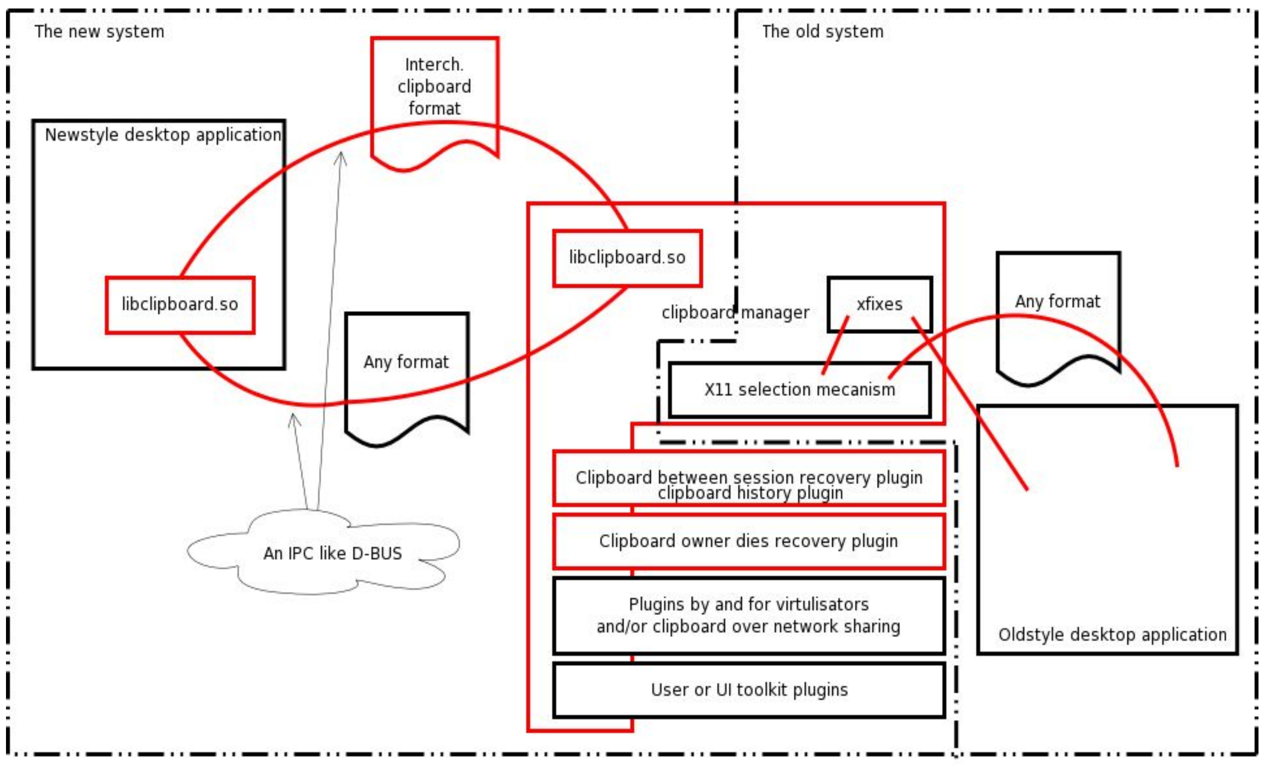
> When an older X11 application requests a target that isn't in use by this new clipboard system, the handler for that request in the clipboard manager can connect with the application that's owning the new-style clipboard. It can then convert it or ask for the same format which the old-style application requested. Now it can transfer/convert it using the older X11 protocol.
>
> This is of course just an implementation idea. And only one direction (I could easily describe the other direction too).

Architectural overview

This overview isn't showing a normal clipboard transfer. The overview is showing for example how backwards compatibility could be addressed (using a clipboard manager).

Red means "*this is something that needs to be developed*".

Black means "*this is something that's already available or isn't important yet*".


The new system
The old system
Interch. clipboard format
Newstyle desktop application
libclipboard.so
libclipboard.so
Any format
clipboard manager
xfixes
X11 selection mecanism
Any format
Clipboard between session recovery plugin
clipboard history plugin
Clipboard owner dies recovery plugin
Plugins by and for virtulisators and/or clipboard over network sharing
User or UI toolkit plugins
An IPC like D-BUS
Oldstyle desktop application

Acknowledgements

I'd like to thank Ms. Tinne Hannes <tinne.hannes@telenet.be> for her linguistical support while she was limited in time due her school exams.

References

[1] X Selections, Cut Buffers, and Kill Rings
http://www.jwz.org/doc/x-cut-and-paste.html
1997, 2002 by Jamie Zawinski

[2] Clipboard specifications
http://standards.freedesktop.org/clipboards-spec/clipboards-0.1.txt
2001 by Havoc Pennington

[3] Clipboard Extensions
http://cvs.freedesktop.org/*checkout*/icccm-extensions/selections/clipboard-extensions.txt?rev=HEAD
2004 by Heinrich Wendel

[4] Peer-to-Peer Communication by Means of Selections
http://tronche.com/gui/x/icccm/sec-2.html#s-2
ICCCM

[5] Xslt transformations
http://www.w3.org/TR/xslt
1999 by James Clark, W3C

[6] Extensible Markup Language (XML)
http://www.w3.org/XML/
W3C

[7] Clipboards in Windows
http://mail.gnome.org/archives/desktop-devel-list/2003-September/msg00257.html
2003 by Mike Hearn