Remote Control and Scripting of Gnome Applications with Python

linux.conf.au 2004
Adelaide, Australia

James Henstridge <james@daa.com.au>

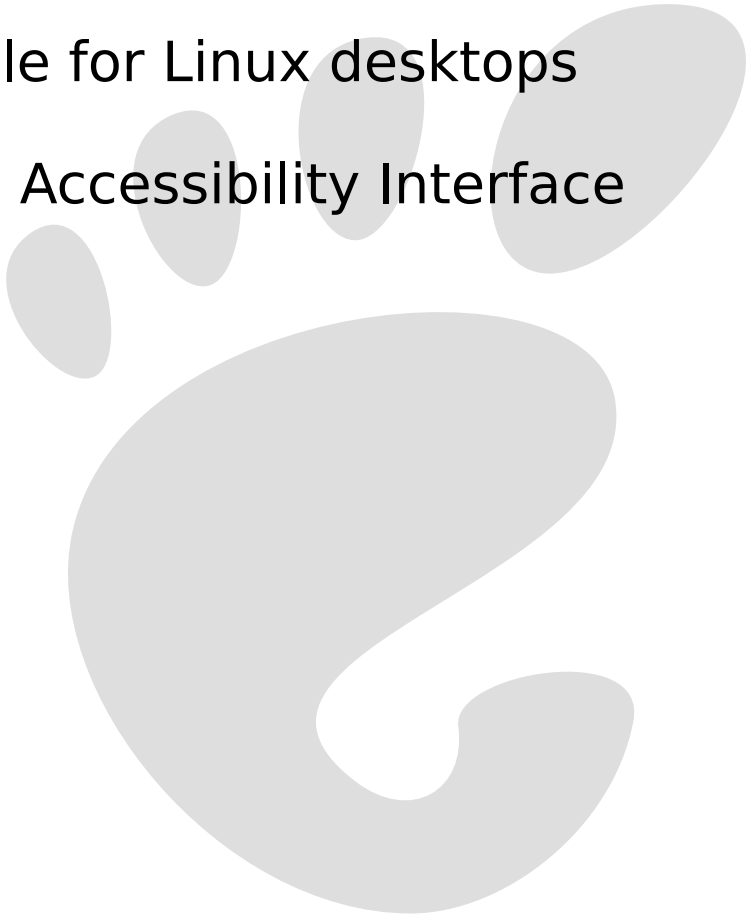
Introduction

What can be done on Windows

Possible solutions available for Linux desktops

Scripting with the Gnome Accessibility Interface

Future directions.



Remote Control on Windows

Many large applications expose their document model via COM

- unified interface for in-process and out of process method invocations.

Once a language has a COM binding, it has access to the COM objects on the machine.

- no need for a specialised language specific interface for each application.
- apps only need to expose their interfaces once.

Real power due to its ubiquity

Remote Control of GUI Apps on Linux

No ubiquitous standard like COM

There are KDE allows some scriptability with DCOP

Some Gnome apps have CORBA interfaces for some of their functionality.

- not many apps do this, and not much is exposed.

Gnome Accessibility framework provides an other way to script applications.

- controls the user interface rather than the data model though
- Gives some level of control over all many apps

Why is the Accessibility Framework Interesting?

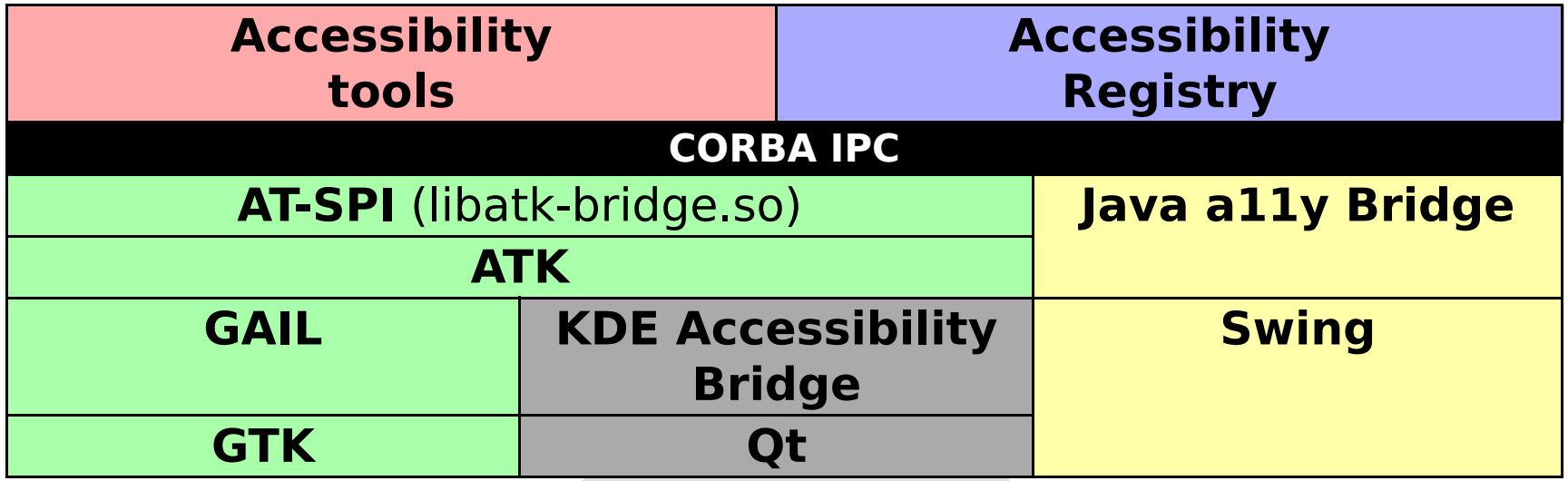
An accessibility tool is used in place of the standard input or output of a program.

- Needs to understand what the UI is displaying.
- Needs to get notification of various application events.
  - focus changes, keyboard input, window manager events, etc.
- Generate keyboard and mouse events.

Accessibility tool requirements overlap with those of remote control or scripting.

There are laws in various countries requiring government purchases of software to favour accessible solutions, so it is in the best interest of companies selling Gnome to make sure it works.

Gnome Accessibility Stack



Gnome Accessibility Stack (continued)

ATK

- In process.
- Maintains a tree of objects that describe the user interface.
- Application authors can add information to this tree.

libatk-bridge.so

- In process.
- Implements the AT-SPI CORBA interfaces, exposing the information ATK maintains.

Accessibility Registry

- Accessible applications register with the registry.
- Allows a11y tools to find applications.
- Allows a11y tools to register global event listeners.

Enabling the Accessibility Framework

By default, the Gnome accessibility framework is disabled.

In order to turn it on, from the Applications menu, choose **Desktop Preferences**, then **Accessibility** and then **Assistive Technology Preferences**. Check the "enable assistive technologies" check box and then log out and back in again.

Applications should now provide the AT-SPI interfaces and connect to the accessibility registry.

Implementation Details

AT-SPI CORBA interface is made up of Bonobo components.

A11y tools find the registry using Bonobo activation.

Scripting languages with CORBA bindings can make use of the framework with little or no extras.

- Python will be used in the examples, making use of PyORBit.

Bonobo

Bonobo is a component system that draws ideas from COM.

Components are made up of collections of interfaces tied together using aggregation

- As opposed to being a single object using multiple inheritance.

Each interface on the component derives from `Bonobo::Unknown`

`Bonobo::Unknown` consists of three methods:

- `ref()` and `unref()` for memory management.
- `queryInterface()` for discovering other interfaces in the component.

Bonobo Activation

The Bonobo activation framework is used to find server implementations based on the results of queries.

Queries can be performed based on:

- the unique identifier of a component implementation.
- what interfaces the component implements.
- arbitrary properties associated with the component.
  - eg. human readable name, description, supported mime types.

If the matching component isn't currently running, Bonobo activation can activate a new instance.

- For out of process components, the component will be forked and executed.
- For in process components, the component is opened with `dlopen()` and constructed.

Scripting Languages

Rather than needing special support for the accessibility framework, a scripting language only needs a CORBA binding and be able to activate components with Bonobo activation.

Python can be used for this purpose using the **pyorbit** and **gnome-python** packages.

Unlike some Python ORBs, pyorbit takes advantage of ORBit's introspection features, allowing it to load up interfaces at runtime. This can happen in one of three ways:

- a type library can be loaded, which contains information about a related set of interfaces.
- IDL files can be parsed at runtime to create interface info.
- If you connect to a server implemented with ORBit, pyorbit can download the interface info from it.

Accessibility Data Model

To use the accessibility framework, you first need to get a reference to the registry. This can be done by activating the identifier `OAFIID:Accessibility_Registry:1.0`.

The registry's `getDesktop()` method can be used to get access to the desktop(s) in the login session.

The desktop object is the root of a tree of `Accessible` objects, with applications as the children.

Application accessibles have window frames as children.

Accessibles

Accessible objects provide the following:

- A name and description
- The "role" of the accessible.
- A list of child accessibles. Can be accessed using the `getChildAtIndex()` method.
- Relationships to other accessibles.
- A set of currently active states for the accessible.

In addition, an accessible may implement additional accessibility interfaces. The role attribute may indicate that the accessible supports certain other interfaces.

Specialised Interfaces

`queryInterface()` can be used to discover the additional interfaces of the accessible.

Action	Actions that can be invoked on the accessible.
Component	A "widget"
Text and EditableText	Text contained in the accessible.
Hypertext	URLs found in the accessible.
Image	Image information.
Selection	Option list style behaviour
Stream	Retrieve content in various formats.
Table	A table of other accessibles.
Value	A range value.

Events

The accessibility registry can also be used to listen for and create events.

The `registerGlobalEventListener()` method can be used to register an event listener object to receive events of a particular type. When such an event occurs, the listener's `notifyEvent()` method will be invoked.

- events include "`focus:`", "`mouse:button`", "`object:state-changed`", "`object:text-changed`".

To generate keyboard and mouse events, we call the registry's `getDeviceEventController()` method to get an `Accessible::DeviceEventController` object.

The `generateKeyboardEvent()` and `generateMouseEvent()` methods on this object are used to synthesise the events.

Using the Accessibility Framework from Python

The following is the boilerplate code needed to start using the a11y interfaces.

```
import ORBit
import bonobo, gnome

ORBit.load_typelib('Accessibility')
gnome.init('list-apps', '0.0')

REGISTRY_IID = 'OAFIID:Accessibility_Registry:1.0'
registry = bonobo.activation.activate("iid == '%s'" % REGISTRY_IID)
```

We can then get a reference to the desktop with:

```
desktop = registry.getDesktop(0)
```

Accessibility Framework and Python (continued)

Iterating over the children of an accessible is trivial. The following code lists the currently running accessible applications:

```
for i in range(desktop.childCount):
    child = desktop.getChildAtIndex(i)
    print child.name
    child.unref()
```

If we have an accessible corresponding to an entry field, queryInterface() can be used to use the `Accessibility::Text` methods on it:

```
text = accessible.queryInterface('IDL:Accessible/Text:1.0')
print text.getText(0, text.characterCount)
```

Event Listeners

Listening for events requires the creation of a CORBA servant:

```
import Accessibility__POA
class MyListener(Accessibility__POA.EventListener):
    def ref(self): pass
    def unref(self): pass
    def queryInterface(self, repo_id):
        if repo_id == 'IDL:Accessibility/EventListener:1.0':
            return self._this()
        else:
            return None

    def notifyEvent(self, event):
        print event.source.getRoleName()

listener = MyListener()
objref = listener._this()
listener._default_POA().the_POAManager.activate()
```

Examples

list-apps.py
Lists all the accessible apps currently running.

dump-tree.py
dumps the accessible tree for a particular app.

follow-focus.py
Prints information about the currently focused application.

spell-check.py
suggests corrections for mis-spellings found in text accessibles.

open-nautilus.py
performs the "open" aciton on the first icon found.

Other Ideas

Use the `Accessibility::EditableText` interface to provide a universal spelling correction program, rather than simply identifying mistakes.

Monitor hyperlinks in focused accessible, and suggest actions (similar to klipper and the selection).

Test suites for the GUI of applications.

Conclusion

Will we eventually get a desktop-neutral scripting system?

- D-BUS might be the answer.

For full document model scripting, things need to be easy to implement.

- Implementing CORBA interfaces in C is non trivial.
- COM has the benefit of being ubiquitous on Windows, so the cost/benefits ratio is different.