Paying technical debt in our accessibility infrastructure

Federico Mena Quintero
(he/him)
federico@gnome.org
@federicomena@mstdn.mx

GUADEC 2022
at-spi-registryd

atk-adaptor

libatspi

pyatspi2

gtk3

firefox

libreoffice

atk

orca

accerciser

gtk4

webkit

odilia

gnome-shell

qt5

toolkits

assistive tech

in-process

context switch
atk:

Just interfaces.

Toolkits (st in gnome-shell) implement them.

“List children”

“Get accessible name”

“Get accessible role”

“Run action”
atk-adaptor:

Translate from atk’s representation to the DBus API’s representation.

Cache some things so toolkit doesn’t need to be asked again for them.

Implement the DBus interfaces in terms of calls to atk.

Talk via DBus.
libatspi:

Talk via DBus.

Convert from DBus APIs to internal representation.

List of accessible objects.

Cached properties for objects.
pyatspi2:
“Old API in Python”
“New API” (GObject Introspection)
Just a language binding for libatspi.
Orca:
A screenreader.
120K lines of Python (!)
Uses pyatspi2’s “Old API”
Awful circumstances?

We think awful code is written by awful devs. But in reality, it's written by reasonable devs in awful circumstances.

https://twitter.com/sarahmei/status/893237308316565505
1997-2018

- cvs.gnome.org - 1997
- bugzilla.gnome.org - 1998
- Tara Hernandez invents CI at Mozilla - 1998 (Tinderbox):

- 2001-2002 - Sun contributes accessibility for GNOME 2
  "Archaeology of Accessibility" (Emmanuele Bassi, GUADEC 2020):
  https://lwn.net/Articles/826738/
  https://www.youtube.com/watch?v=eNh0Xg8abj0

- svn.gnome.org - 2006?
- gtestutils (glib) - 2007
- git.gnome.org - 2008

- Oracle acquires Sun; Accessibility team disappears (2010-2011)

- GitHub - 2008; Travis CI - 2011
- containers - Jessie Frazelle and Docker - 2013 - 2015?
- rootless containers - 2016
- gitlab.gnome.org - 2018
Typical state of legacy code

- **Librsvg:** few tests which don’t really work, no CI, no reproducible environment.
- **Accessibility:** few tests which don’t really work, no CI, no reproducible environment.
- **Yelp:** few tests which don’t really work, no CI, no reproducible environment.
- **gnome-session:** few tests which don’t really work, no CI, no reproducible environment.
Lines of code in at-spi2-core

2009-2010: CORBA → DBus
Split modules

https://github.com/erikbernh/git-of-theseus
First: add CI

• Copied .gitlab-ci.yml from libgweather (thanks, ebassi!)

• Needed a Meson project in C with nice things: static analysis, address sanitizer, gi-docgen, test coverage reports.

• Replaced hand-written Fedora setup with Freedesktop CI Templates (copied from librsvg - thanks, alatiera!)
<table>
<thead>
<tr>
<th>Container-build</th>
<th>Build</th>
<th>Analysis</th>
<th>Docs</th>
<th>Deploy</th>
</tr>
</thead>
<tbody>
<tr>
<td>fedora-container@x86_64</td>
<td>fedora-x86_64</td>
<td>asan-build</td>
<td>reference</td>
<td>pages</td>
</tr>
<tr>
<td>opensuse-container@x86_64</td>
<td>opensuse-x86_64</td>
<td>coverage</td>
<td>static-scan</td>
<td>pages:deploy</td>
</tr>
</tbody>
</table>
Build a reproducible environment (container image)

Automatically uploads it to gitlab.gnome.org’s container registry

No “run-docker.sh” to be run by hand on maintainer’s computer

“Instant” if container has not changed between commits

Freedesktop CI Templates: https://gitlab.freedesktop.org/freedesktop/ci-templates/
Compile and run tests

Compiles code

Runs the tests (openSUSE only / dbus-daemon)

HELP WANTED for Fedora / dbus-broker / needs a VM?
Modern compilers are awesome

Address sanitizer - for a memory-unsafe language

Static analysis - because compilers are way better than in 2005

Test coverage reports - “what code was executed?”
Generate HTML, publish it

Build documentation

Publish documentation and HTML code coverage report to Gitlab pages
Add a CI pipeline
Add a CI pipeline

Add lots of little tests
Add a CI pipeline

Add lots of little tests

Become the CI person
Add a CI pipeline

Add lots of little tests

Become the CI person

Become the CI person
<table>
<thead>
<tr>
<th>Directory</th>
<th>Line Coverage</th>
<th>Functions</th>
<th>Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>atk</td>
<td>55.64%</td>
<td>62.23%</td>
<td>33.27%</td>
</tr>
<tr>
<td>atk-adaptor</td>
<td>42.5%</td>
<td>46.67%</td>
<td>22.59%</td>
</tr>
<tr>
<td>atk-adaptor/adaptors</td>
<td>60.75%</td>
<td>77.03%</td>
<td>36.13%</td>
</tr>
<tr>
<td>atk-adaptor/gtk-2.0</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>atspi</td>
<td>53.18%</td>
<td>62.82%</td>
<td>35.26%</td>
</tr>
<tr>
<td>bus</td>
<td>51.59%</td>
<td>60%</td>
<td>32.47%</td>
</tr>
<tr>
<td>dbind</td>
<td>70.92%</td>
<td>85.29%</td>
<td>49.26%</td>
</tr>
<tr>
<td>droute</td>
<td>57%</td>
<td>61.7%</td>
<td>41.18%</td>
</tr>
<tr>
<td>registryd</td>
<td>22.65%</td>
<td>27.62%</td>
<td>15.08%</td>
</tr>
<tr>
<td>tests/at-spi2-atk</td>
<td>98.74%</td>
<td>99.53%</td>
<td>54.28%</td>
</tr>
<tr>
<td>tests/at-spi2-atk/dummyatk</td>
<td>85.58%</td>
<td>77.11%</td>
<td>52.12%</td>
</tr>
<tr>
<td>tests/atk</td>
<td>89.31%</td>
<td>88.24%</td>
<td>43.48%</td>
</tr>
<tr>
<td>tests/atspi</td>
<td>53.49%</td>
<td>71.43%</td>
<td>42.31%</td>
</tr>
</tbody>
</table>
### Coverage report for librsvg

<table>
<thead>
<tr>
<th>Directory</th>
<th>Line Coverage</th>
<th>Functions</th>
<th>Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Line Coverage</td>
<td>Functions</td>
<td>Branches</td>
</tr>
<tr>
<td>examples</td>
<td>0%</td>
<td>0%</td>
<td>100%</td>
</tr>
<tr>
<td>gdk-pixbuf-loader</td>
<td>12.61%</td>
<td>22.22%</td>
<td>0%</td>
</tr>
<tr>
<td>src</td>
<td>95.4%</td>
<td>37.53%</td>
<td>100%</td>
</tr>
<tr>
<td>src/bin</td>
<td>96.98%</td>
<td>37.82%</td>
<td>100%</td>
</tr>
<tr>
<td>src/c_api</td>
<td>8.64%</td>
<td>6.32%</td>
<td>100%</td>
</tr>
<tr>
<td>src/filters</td>
<td>93.97%</td>
<td>34.29%</td>
<td>100%</td>
</tr>
<tr>
<td>src/surface_utils</td>
<td>89.98%</td>
<td>38.28%</td>
<td>100%</td>
</tr>
<tr>
<td>src/xml</td>
<td>86.36%</td>
<td>48.1%</td>
<td>100%</td>
</tr>
<tr>
<td>tests</td>
<td>94.87%</td>
<td>100%</td>
<td>50%</td>
</tr>
<tr>
<td>tests/src</td>
<td>97.88%</td>
<td>97.67%</td>
<td>100%</td>
</tr>
<tr>
<td>tests/src/cmdline</td>
<td>99.89%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>tests/src/predicates</td>
<td>58.93%</td>
<td>60.2%</td>
<td>100%</td>
</tr>
</tbody>
</table>

**Overall Coverage:**
- Lines: 88.73%
- Functions: 44.81%
- Branches: 47.01%
## Coverage report for glib

**Current view:** top level  
**Test:** unnamed  
**Date:** 2022-07-15 22:27:23

<table>
<thead>
<tr>
<th>Directory</th>
<th>Line Coverage</th>
<th>Functions</th>
<th>Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>fuzzing</td>
<td>85.4 %</td>
<td>100.0 %</td>
<td>48.4 %</td>
</tr>
<tr>
<td>gio</td>
<td>64.5 %</td>
<td>76.2 %</td>
<td>50.8 %</td>
</tr>
<tr>
<td>gio/inotify</td>
<td>72.2 %</td>
<td>84.5 %</td>
<td>56.0 %</td>
</tr>
<tr>
<td>gio/tests</td>
<td>84.9 %</td>
<td>84.2 %</td>
<td>56.9 %</td>
</tr>
<tr>
<td>gio/tests/modules</td>
<td>60.0 %</td>
<td>57.1 %</td>
<td>50.0 %</td>
</tr>
<tr>
<td>gio/win32</td>
<td>11.9 %</td>
<td>16.7 %</td>
<td>11.1 %</td>
</tr>
<tr>
<td>gio/xdg_mime</td>
<td>36.3 %</td>
<td>47.3 %</td>
<td>27.3 %</td>
</tr>
<tr>
<td>glib</td>
<td>83.9 %</td>
<td>95.0 %</td>
<td>71.8 %</td>
</tr>
<tr>
<td>glib/deprecated</td>
<td>78.8 %</td>
<td>68.7 %</td>
<td>67.7 %</td>
</tr>
<tr>
<td>glib/gnulib</td>
<td>50.6 %</td>
<td>71.9 %</td>
<td>39.7 %</td>
</tr>
<tr>
<td>glib/libcharset</td>
<td>48.7 %</td>
<td>100.0 %</td>
<td>29.6 %</td>
</tr>
<tr>
<td>glib/tests</td>
<td>96.1 %</td>
<td>97.6 %</td>
<td>66.0 %</td>
</tr>
<tr>
<td>glib/tests/path-test-subdir</td>
<td>100.0 %</td>
<td>100.0 %</td>
<td>0 / 0</td>
</tr>
<tr>
<td>gmodule</td>
<td>69.6 %</td>
<td>76.5 %</td>
<td>53.8 %</td>
</tr>
<tr>
<td>gmodule/tests</td>
<td>86.7 %</td>
<td>100.0 %</td>
<td>48.0 %</td>
</tr>
<tr>
<td>gobj ect</td>
<td>81.2 %</td>
<td>90.4 %</td>
<td>64.8 %</td>
</tr>
<tr>
<td>gobj ect/tests</td>
<td>92.8 %</td>
<td>91.4 %</td>
<td>60.4 %</td>
</tr>
<tr>
<td>gobj ect/tests/performance</td>
<td>86.0 %</td>
<td>93.7 %</td>
<td>66.0 %</td>
</tr>
<tr>
<td>gthread</td>
<td>0.0 %</td>
<td>0.0 %</td>
<td>0.0 %</td>
</tr>
</tbody>
</table>
static DBusMessage *
impl_GetChildren (DBusMessage * message, SpiRegistry *registry)
{
    DBusMessage *reply = NULL;
    DBusMessageIter iter, iter_array;
    int i;
    reply = dbus_message_new_method_return (message);
    dbus_message_iter_init_append (reply, &iter);
    dbus_message_iter_open_container(&iter, DBUS_TYPE_ARRAY, "(so)", &iter_array);
    for (i=0; i < registry->apps->len; i++)
    {
        SpiReference *current = g_ptr_array_index (registry->apps, i);
        append_reference (&iter_array, current->name, current->path);
    }
    dbus_message_iter_close_container(&iter, &iter_array);
    return reply;
}

static DBusMessage *
impl_GetIndexInParent (DBusMessage * message, SpiRegistry *registry)
{
    DBusMessage *reply;
    dbus_uint32_t rv = 0;
    reply = dbus_message_new_method_return (message);
    dbus_message_append_args (reply, DBUS_TYPE_INT32, &rv, DBUS_TYPE_INVALID);
    return reply;
}
def test_accessible_iface_properties(registry_root, session_manager):
    values = [
        ('Name', 'main'),
        ('Description', ''),
        ('Parent', '', '/org/a11y/atspi/null'),
        ('ChildCount', 0),
    ]

    for prop_name, expected in values:
        assert get_property(registry_root, ACCESSIBLE_IFACE, prop_name) == expected

    if (!strcmp (prop_iface, SPI_DBUS_INTERFACE_ACCESSIBLE))
        {
            if (!strcmp (prop_member, "Name"))
                impl_get_Name (&iter, registry);
            else if (!strcmp (prop_member, "Description"))
                impl_get_Description (&iter, registry);
            else if (!strcmp (prop_member, "Parent"))
                impl_get_Parent (&iter, registry);
            else if (!strcmp (prop_member, "ChildCount"))
                impl_get_ChildCount (&iter, registry);
            else
                {
                    dbus_message_unref (reply);
                    reply = dbus_message_new_error (message, DBUS_ERROR_FAILED, "Property unavailable");
                }
        }
def test_accessible_iface_properties(registry_root, session_manager):
    values = [
        ('Name', 'main'),
        ('Description', ''),
        ('Parent', ('', '/org/a11y/atspi/null'),
        ('ChildCount', 0),
    ]
    for prop_name, expected in values:
        assert get_property(registry_root, ACCESSIBLE_IFACE, prop_name) == expected

def test_unknown_property_yields_error(registry_root, session_manager):
    with pytest.raises(dbus.exceptions.DBusException):
        get_property(registry_root, ACCESSIBLE_IFACE, 'NonexistentProperty')

if (!strcmp (prop_iface, SPI_DBUS_INTERFACE_ACCESSIBLE))
{
    if (!strcmp (prop_member, "Name")
        impl_get_Name (&iter, registry);
    else if (!strcmp (prop_member, "Description")
        impl_get_Description (&iter, registry);
    else if (!strcmp (prop_member, "Parent")
        impl_get_Parent (&iter, registry);
    else if (!strcmp (prop_member, "ChildCount")
        impl_get_ChildCount (&iter, registry);
    else
    {
        dbus_message_unref (reply);
        reply = dbus_message_new_error (message, DBUS_ERROR_FAILED, "Property unavailable");
    }
}
Coverage in merge request diffs

Executed (green)

Not executed (red)
The incantation for coverage in diffs

artifacts:
  name: "at-spi2-core-\${CI_JOB_NAME}-\${CI_COMMIT_REF_NAME}"
  expire_in: 2 days
  when: always
  reports:
    coverage_report:
      coverage_format: cobertura
      path: coverage.xml

grcov _build --source-dir ./ --prefix-dir ../ --output-type cobertura --branch \
  --ignore-not-existing -o coverage.xml
```c
dbus_message_iter_close_container(&iter, &iter_array);
return reply;
}

static DBusMessage *
impl_GetIndexInParent (DBusMessage * message, SpiRegistry *registry)
{
    DBusMessage *reply;
    dbus_uint32_t rv = 0;
```
Important detail about coverage

- Your processes must exit cleanly!!!!!!

- gcc/clang’s runtimes write the coverage info at exit.

- Terminate with SIGTERM / SEGV / etc, no coverage info gets written.
Daemons: bus-launcher and registryd

- Their lifetime is controlled by the session manager.
- Had no session manager during test suite.
- Wrote a mock gnome-session, 80 lines.
- Martin Pitt’s python-dbusmock is awesome!
pytest

• Fixture is “an accessibility registry daemon tied to the session manager”.

• Test test test, all via DBus in Python.

• Fixture teardown is “tell the mock session manager to Logout”.

• This terminates the daemons cleanly → get coverage.
@pytest.fixture
def session_manager():
    bus = dbus.SessionBus()
    mock_session = bus.get_object('org.gnome.SessionManager', '/org/gnome/SessionManager')

    yield mock_session

    # Tell all session clients to terminate
    mock_session.Logout(0, dbus_interface='org.gnome.SessionManager')
static DBusMessage*
impl_Embed (DBusMessage *message, SpiRegistry *registry)
{
    DBusMessageIter iter, iter_struct;
    const gchar *app_name, *obj_path;

    DBusMessage *reply = NULL;
    DBusMessageIter reply_iter;
    SpiReference *app_root;

    dbus_message_iter_init (message, &iter);
    dbus_message_iter_recurse (&iter, &iter_struct);
    if (!dbus_message_iter_get_arg_type (&iter_struct) == DBUS_TYPE_STRING)
        goto error;
    dbus_message_iter_get_basic (&iter_struct, &app_name);
    if (!app_name)
        app_name = dbus_message_get_sender (message);
    if (!dbus_message_iter_next (&iter_struct))
        goto error;
    if (!dbus_message_iter_get_arg_type (&iter_struct) == DBUS_TYPE_OBJECT_PATH)
        goto error;
    dbus_message_iter_get_basic (&iter_struct, &obj_path);

    app_root = spi_reference_new (app_name, obj_path);
    add_application (registry, app_root);

    set_id (registry, app_root);

    reply = dbus_message_new_method_return (message);
    dbus_message_iter_init_append (reply, &reply_iter);
    append_reference (&reply_iter,
        registry->bus_unique_name,
        SPI_DBUS_PATH_ROOT);

    return reply;
error:
    return dbus_message_new_error (message, DBUS_ERROR_FAILED, "Invalid arguments");
}
static DBusMessage*
impl_Embed (DBusMessage *message, SpiRegistry *registry)
{
    DBusMessageIter iter, iter_struct;
    const gchar *app_name, *obj_path;

    DBusMessage *reply = NULL;
    DBusMessageIter reply_iter;
    SpiReference *app_root;

    dbus_message_iter_init (&iter, message);
    dbus_message_iter_recursively (&iter, &iter_struct);
    if (!dbus_message_iter_get_arg_type (&iter_struct) == DBUS_TYPE_STRING))
        goto error;
    dbus_message_iter_get_basic (&iter_struct, &app_name);
    if (!app_name
        app_name = dbus_message_get_string (message);
    if (!dbus_message_iter_next (&iter_struct))
        goto error;
    if (!dbus_message_iter_get_arg_type (&iter_struct) == DBUS_TYPE_OBJECT_PATH))
        goto error;
    dbus_message_iter_get_basic (&iter_struct, &obj_path);

    app_root = spi_reference_new (app_name, obj_path);
    add_application (registry, app_root);

    set_id (registry, app_root);

    reply = dbus_message_new_method_return (message);
    dbus_message_iter_init_append (reply, &reply_iter);
    append_reference (&reply_iter,
        registry->bus_unique_name,
        SPI_DBUS_PATH_ROOT);

    return reply;

error:
    return dbus_message_new_error (message, DBUS_ERROR_FAILED, "Invalid arguments");
}
static DBusMessage*
impl_Embd (DBusMessage *message, SpiRegistry *registry)
{
    DBusMessageIter iter, iter_struct;
    const gchar *app_name, *obj_path;

    DBusMessage *reply = NULL;
    DBusMessageIter reply_iter;
    SpiReference *app_root;

    dbus_message_iter.Init (message, &iter);
    dbus_message_iter.Recurse (&iter, &iter_struct);
    if (!(dbus_message_iter.Get_arg_type (&iter_struct) == DBUS_TYPE_STRING))
        goto error;
    dbus_message_iter_get_basic (&iter_struct, &app_name);
    if (!app_name)
        app_name = dbus_message_get_sender (message);
    if (!(dbus_message_iter_next (&iter_struct))
        goto error;
    if (!(dbus_message_iter_get_arg_type (&iter_struct) == DBUS_TYPE_OBJECT_PATH))
        goto error;
    dbus_message_iter_get_basic (&iter_struct, &obj_path);

    app_root = spi_reference_new (app_name, obj_path);
    add_application (registry, app_root);

    set_id (registry, app_root);

    reply = dbus_message_new_method_return (message);
    dbus_message_iter_init_append (reply, &reply_iter);
    append_reference (&reply_iter,
        registry->bus_unique_name,
        SPI_DBUS_PATH_ROOT);

    return reply;
error:
    return dbus_message_new_error (message, DBUS_ERROR_FAILED, "Invalid arguments");
}
a bit of actual logic

get arguments from DBus

a bit of actual logic

reply = dbus_message_new_method_return (message);
dbus_message_iter_init_append (reply, &reply_iter);
append_reference (&reply_iter,
        registry->bus_unique_name,
        SPI_DBUS_PATH_ROOT);

return reply;
error:
return dbus_message_new_error (message, DBUS_ERROR_FAILED, "Invalid arguments");
static DBusMessage *
impl_Embed (DBusMessage *message, SpiRegistry *registry)
{
  DBusMessageIter iter, iter_struct;
  const gchar *app_name, *obj_path;

  DBusMessage *reply = NULL;
  DBusMessageIter reply_iter;
  SpiReference *app_root;

  dbus_message_iter_init (message, &iter);
  dbus_message_iter_recurse (&iter, &iter_struct);
  if (!dbus_message_iter_get_arg_type (&iter_struct) == DBUS_TYPE_STRING)
    goto error;
  dbus_message_iter_get_basic (&iter_struct, &app_name);
  if (!app_name)
    app_name = dbus_message_get_sender (message);
  if (!dbus_message_iter_next (&iter_struct))
    goto error;
  if (!dbus_message_iter_get_arg_type (&iter_struct) == DBUS_TYPE_OBJECT_PATH)
    goto error;
  dbus_message_iter_get_basic (&iter_struct, &obj_path);

  app_root = spi_reference_new (app_name, obj_path);
  add_application (registry, app_root);
  set_id (registry, app_root);

  reply = dbus_message_new_method_return (message);
  dbus_message_iter_init_append (reply, &reply_iter);
  append_reference (&reply_iter,
    [SPI_REGISTRY_OBJECT_PATH],
    SpiRegistry::name_to_path(registry),
    app_root);

  return reply;
}

error:
  return dbus_message_new_error (message, DBUS_ERROR_FAILED, "Invalid arguments");
static SpiReference *
socket_embed (SpiRegistry *registry, SpiReference *app_root)
{
  add_application (app_root);
  set_id (registry, app_root);
  return spi_reference_new (registry->bus_unique_name, SPI_DBUS_PATH_ROOT);
}

static DBusMessage*
impl_Embd (DBusMessage *message, SpiRegistry *registry)
{
  SpiReference *app_root = NULL;
  SpiReference *result;
  DBusMessage *reply = NULL;
  DBusMessageIter reply_iter;

  if (socket_embed_demarshar (message, &app_root) != DEMARSHAL_STATUS_SUCCESS)
  {
    return dbus_message_new_error (message, DBUS_ERROR_FAILED, "Invalid arguments");
  }

  result = socket_embed (registry, app_root); /* takes ownership of the app_root */

  reply = dbus_message_new_method_return (message);
  dbus_message_iter_init_append (reply, &reply_iter);
  append_reference (&reply_iter, result->name, result->path);
  spi_reference_free (result);

  return reply;
}
Things we have learned over the years

- Reproducible environments for build/test “works on my machine”
- Code coverage for tests.
- Automated builds.
- Continuous integration.
- Let the robots do the boring work.
- Implemented with marge-bot in librsvg.
Thank you!