Unity Scopes

On the Desktop and Phone

Presentation by
James Henstridge
james.henstridge@canonical.com
www.canonical.com
linunx.conf.au, January 2014

CANONICAL

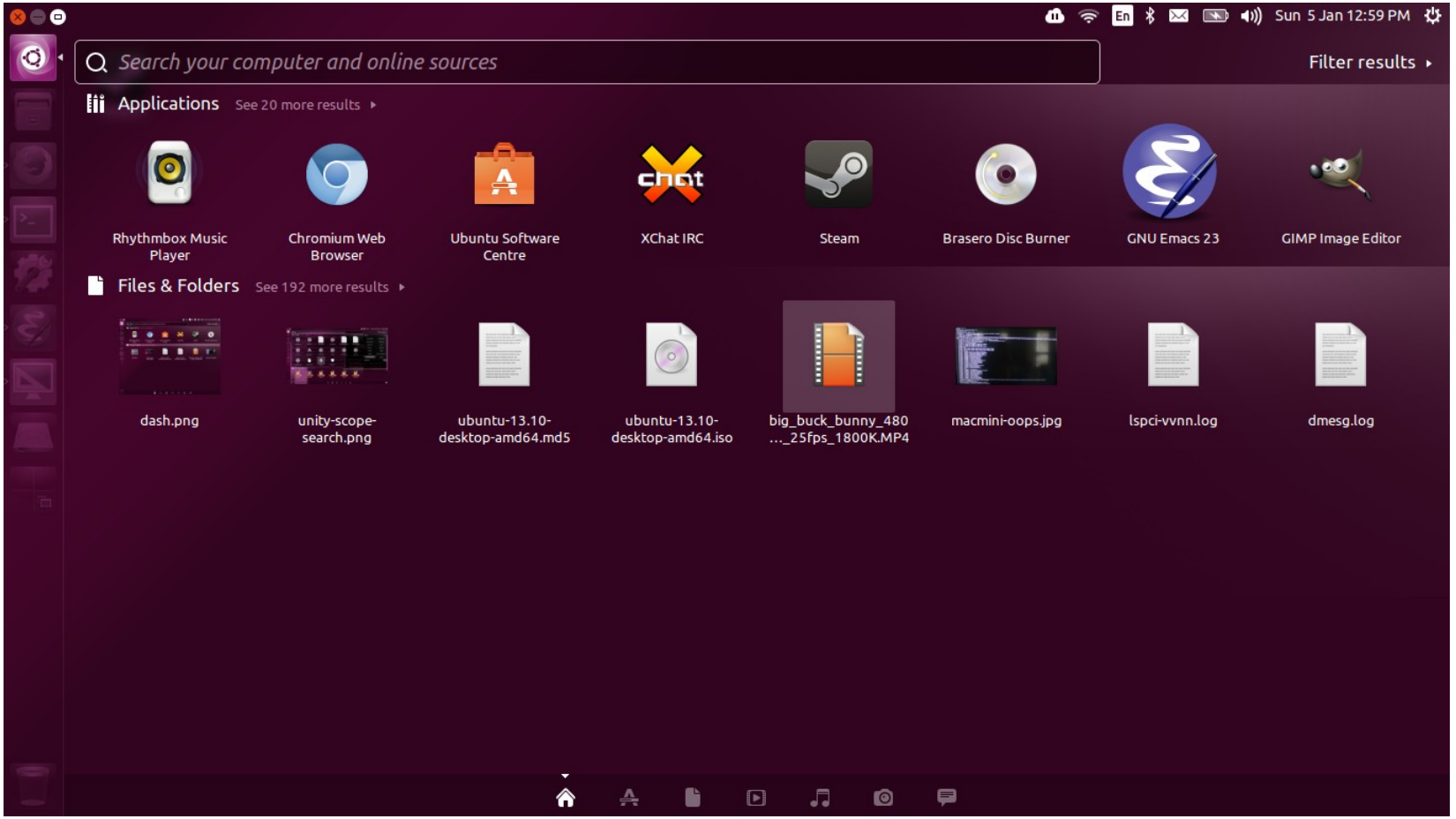
What are scopes?

- Scopes are the back end for the Unity dash (accessed via the Super/Windows key)
- Back end feature set follows requirements of UI
- Provide results for different types of data (applications, files, videos, music, etc)
- Provide results in responses to searches and for “surfacing”
- Determine how to activate or preview results

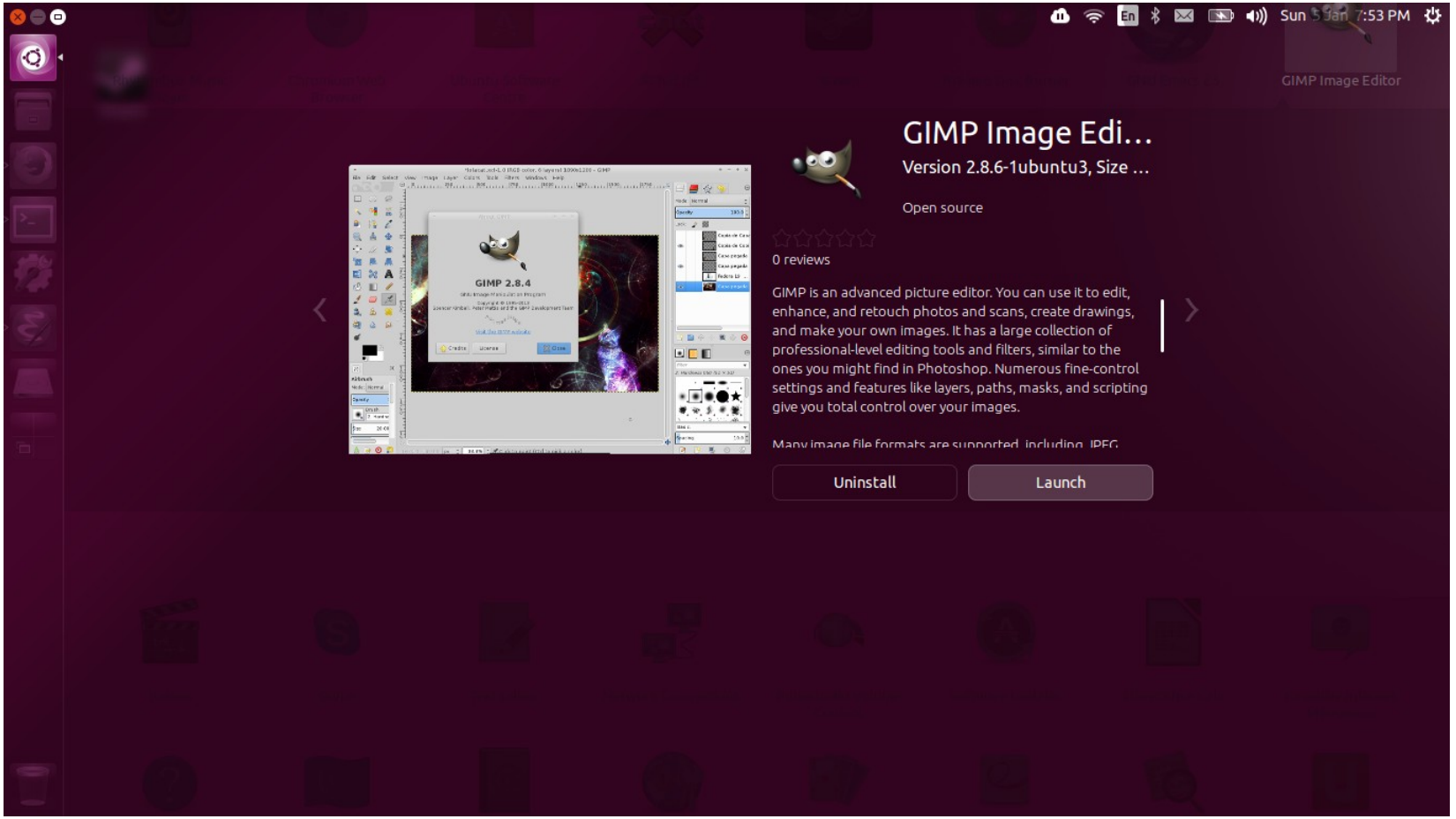
Dash Overview

- Displays a set of “surfacing” results when opened
- Results broken down by category labels
- Incremental search results are displayed as the user types
- Pages for different types of result (apps, files, music, etc) available via tabs at the bottom
- Domain specific filters available to limit results
- Results can be previewed (click), or activated (double click)

Dash Overview (2)



Dash Overview (3)



Results

- A scope result is essentially a collection of metadata key/value items.
- Some items are predefined such as:
  - URI
  - Title
  - Icon
  - Category
- Scopes can also store arbitrary metadata in a result

Searching

- The primary operation for a scope
  - Empty search string used for surfacing
- Results are pushed to the client
- Search can be cancelled, e.g. for incremental searches
- Changes to filters result in new searches

Activating Results

- Client requests that the scope
  - full result dictionary is passed back to the scope
- Scope can reply in a number of ways:
  - NOT_HANDLED: tells client to activate result itself
  - SHOW_DASH/HIDE_DASH: scope handled activation
  - GOTO_PREVIEW: display a preview
  - PERFORM_SEARCH: tell client to perform a new search

Previews

- Slient can request a preview by passing a result to the scope
- Scope can pick one of a small number of templates for the preview (generic, application, music, video, etc)
- One or more action buttons can be attached, which are handled via the activation API

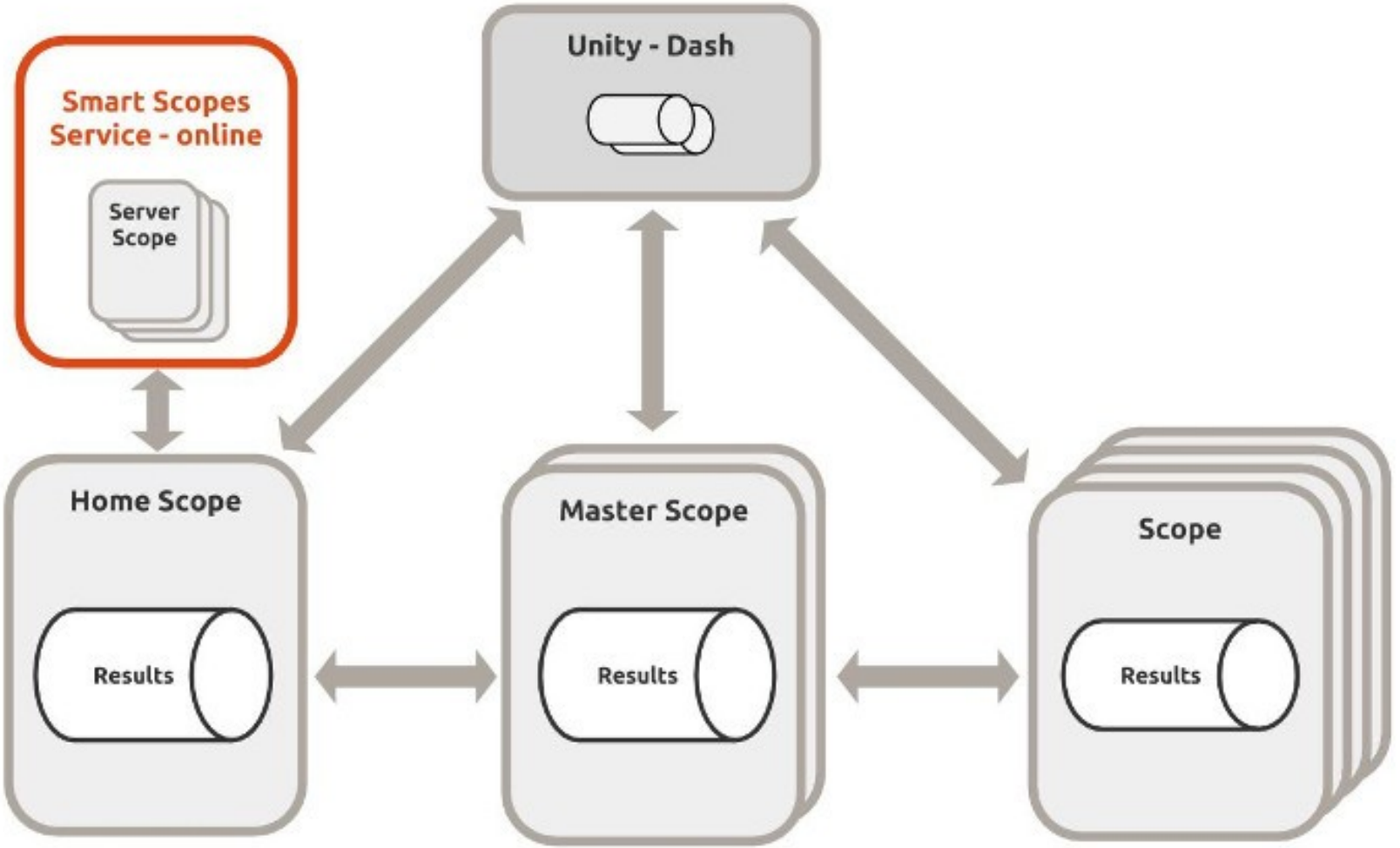
Master Scopes

- Each page of the dash is handled by a master scope
- Master scopes aggregate results from other results
  - provide the same API as regular scopes
- For best the experience, scopes under the same master scope should use the same categories and filters when appropriate
- You probably won't want to write a new master scope: instead, plug in to an existing master scope

Smart Scope Server

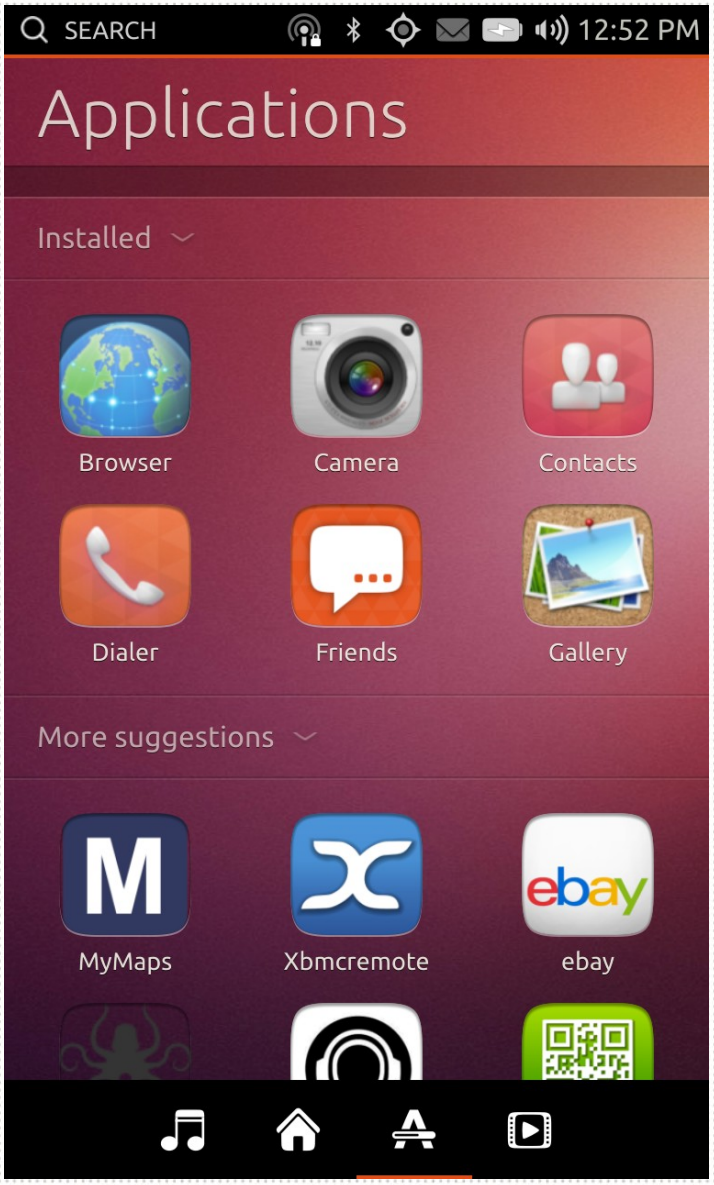
- Scopes run on a remote server, with results sent to client via HTTP
- Access to multiple scopes multiplexed over a single request
- Only suitable for anonymous non-personal results
- Currently integrated via the “home” master scope

Overview



Ref: http://developer.ubuntu.com/scopes/overview/

Phone



- Unity 8 phone shell modelled on the Dash
- Uses the same back end scopes API
- Some scopes needed replacement when they depended on functionality not present on the phone
- Due to limited resources, a push to run as many scopes remotely as possible
- Discourage use of Python for local scopes

Scope Sample

```
from gi.repository import Unity

class MyScope(Unity.AbstractScope):
    def get_categories(self):
        cats = Unity.CategorySet.new()
        cats.add(Unity.Category.new('name', 'Display name',
icon,
            Unity.CategoryRenderer.DEFAULT))
        return cats
    def do_create_search_for_query(self, search_context):
        return MySearch(search_context)
    def do_create_previewer(self, result, metadata):
        return MyPreviewer(result, metadata)
    def do_activate(self, result, metadata):
        ...
```

Scope Sample (2)

```
class MySearch(Unity.ScopeSearchBase):
    def __init__(self, search_context):
        super(MySearch, self).__init__()
        self.set_search_context(search_context)

    def do_run(self):
        query = self.search_context.search_query
        result_set = self.search_context.result_set
        result = Unity.ScopeResult.create(
            uri, icon, category, result_type, mimetype,
            title, comment, dnd_uri, metadata)
        result_set.add_result(result)
        ...
        if self.search_context.cancellable.is_cancelled():
            return
        ...
```

Scope Sample (3)

```
class MyPreviewer(Unity.ResultPreviewer):
    def __init__(self, result, metadata):
        super(MyPreviewer, self).__init__()
        self.set_scope_result(result)
        self.set_search_metadata(metadata)

    def do_run(self):
        preview = Unity.GenericPreview.new(
            self.result.title, '', icon)
        preview.add_action(Unity.PreviewAction.new(
            "open", "Open", None))
        return preview
```

Additional configuration

- Scope configuration file
  - provides basic metadata and IPC endpoint to talk to scope
  - The RemoteContent key used to blanket disable remote access
- configuration file location determines which master scope it feeds data to

Future

- New version of the scopes API
  - C++11 API
  - Go and Javascript bindings in development
- Ability to confine scopes via AppArmor
  - e.g. make “access to network” and “access to personal data” mutually exclusive
  - restrict how scopes can talk with each other

Resources

- Ubuntu Developer website:
  - http://developer.ubuntu.com/scopes/overview/
  - http://developer.ubuntu.com/scopes/tutorial/
- New scopes API:
  - `bzr branch lp:unity-scopes-api`

Questions please

Thank you

James Henstridge
james.henstridge@canonical.com
www.canonical.com