Cake:

A language for composing mismatched binaries

Stephen Kell

What is Cake?

- **Problem**

  Bigger and more complex software projects call for new development practices: better programming languages, decentralised development and unanticipated component re-use. All these entail dealing with **mismatched** interfaces. Glue coding or source-level editing are labour-intensive and yield brittle outputs.

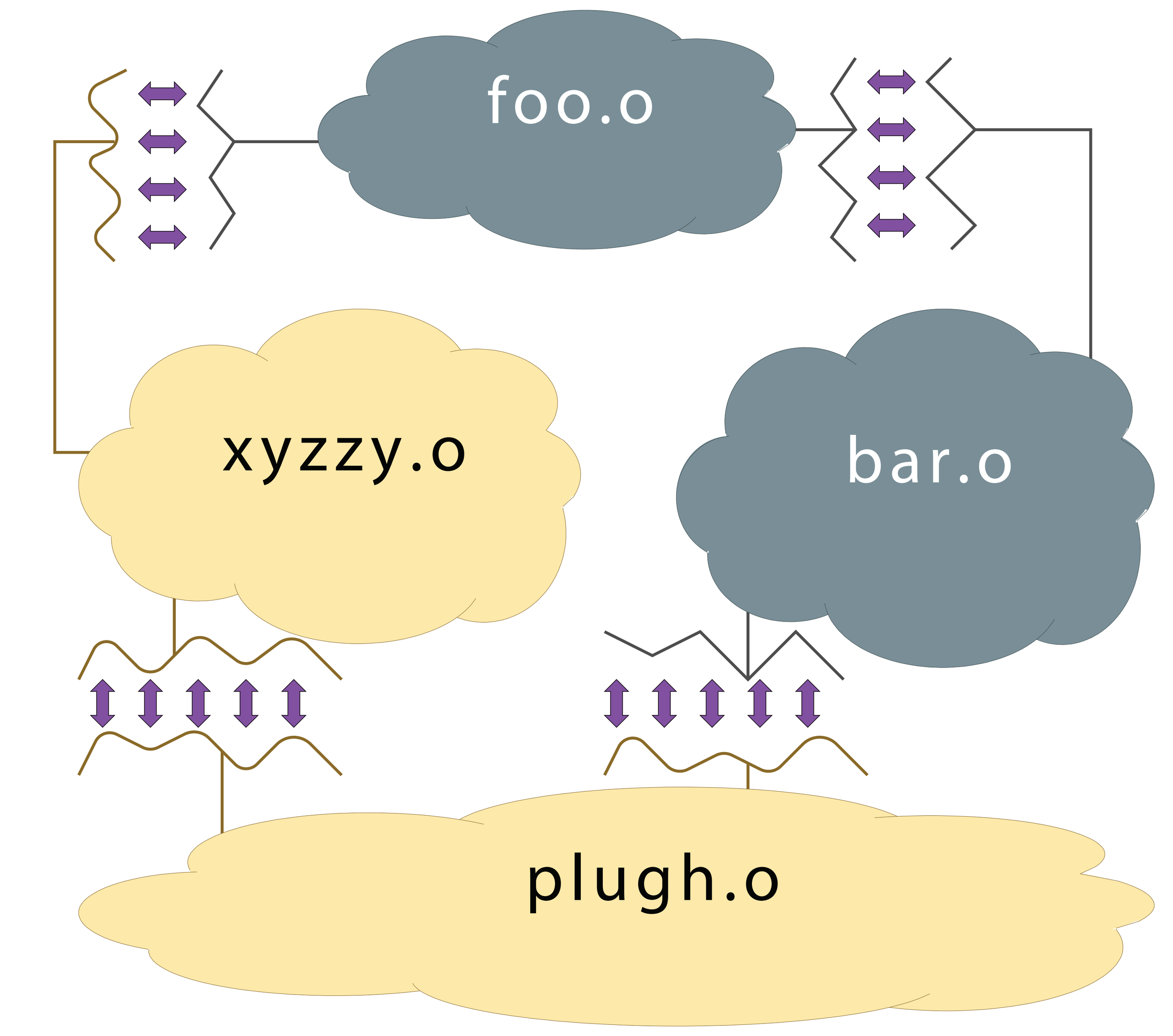
- **Approach**

  Cake is a linking language for composing mismatched **binary** object code. It complements existing languages and toolchains: rather than describing new programmatic artifacts, it expresses **correspondences** between existing ones. By targetting binaries, Cake unifies many source languages.

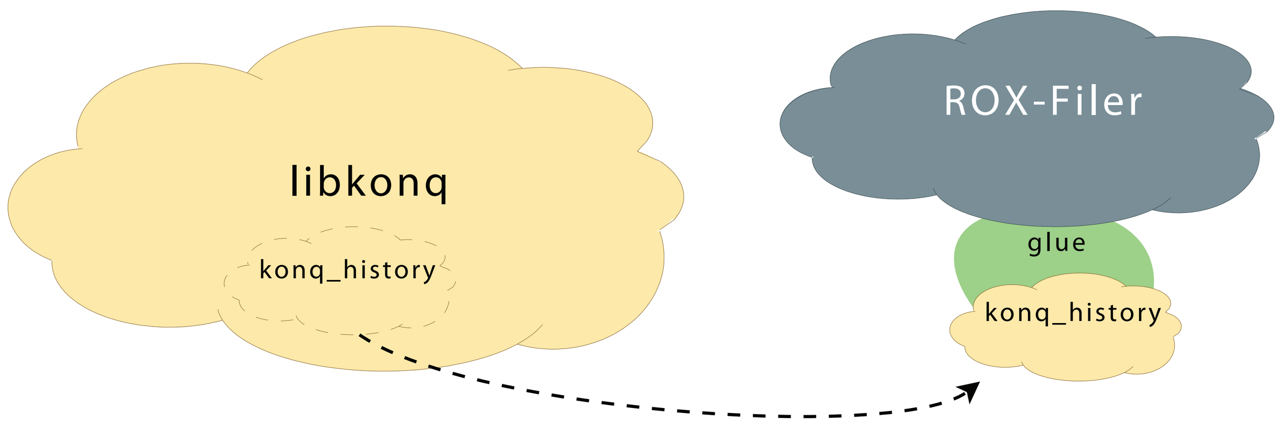
- **Case studies**

  Cake's design is guided by practical case studies. Two have been carried out so far. In the first, to demonstrate **unanticipated composition**, we ported the history feature from the Konqueror browser to run inside the ROX file manager. In the second, to demonstrate **evolution**, we adapted a small Gtk+ client compiled against old library version 1.2 to make it link with new version 2.x.

Big picture and case studies

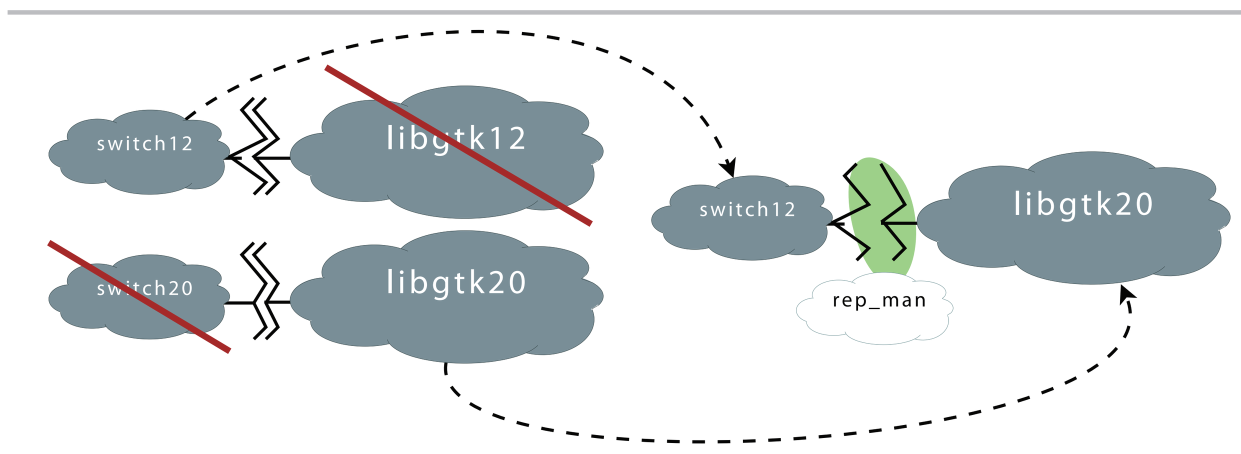


Cake is designed from a model of software as communicating object files. It considers mismatched interfaces in both small-scale details—arguments, names, value structures—and in its large-scale "packaging": the languages and libraries defining the infrastructure and vocabulary on which the interface depends. Both scales of mismatch have been explored in the case studies, where glue coding was performed by hand.



Konqueror + ROX

1. Extricate history feature from containing library
2. Glue to ROX-Filer
3. Initialise state normally set up by Konqueror library



Gtk+ old-with-new

1. Bridge API changes
2. Created *generic* runtime library for exchange of objects with similar content but mismatched layout.

What the language looks like

On the right are two similar but mismatched interfaces. Coloured underlines show the values and functions which correspond between the two.

In Cake, the programmer directly specifies any correspondences not implied by name-matching.

Value correspondences relate structured values.

Function correspondences relate function calls, using a powerful pattern-matching syntax.

```
struct Window {
  Widget super; char * title ; // ...
  WindowType type;
  unsigned window_has_focus:1;
};

void win_set_policy (Window *w,
  bool shrink, bool grow, bool autom);
void handler_connect(Widget *o, char *ev,
  Handler f, void * f_arg );
```

```
struct Window {
  Widget super; char * title ; // ...
  char *wm_role; /* new field */
  unsigned type:4; /* WindowType */
  unsigned has_focus:1;
};

void set_size_request (Widget *w, int w, int h);
void set_resizable (Window *w, bool resizable);
void signal_connect (Object *o, char *event,
    Handler f, void * f_arg, int flags );
```

```
old_client ↔ new_library
{
  values Window ↔ Window {
    const ""                  → wm_role;
    type as WindowType        ↔ type as WindowType;
    window_has_focus          ↔ has_focus;
  }

  handler_connect(w, ev, f, arg)    → signal_connect (w, ev, f, f_arg, {} );

  win_set_policy (w, shrink, grow, _)  →
          ( if shrink then set_size_request (w, 0, 0) else void;
            if grow then set_resizable (w, true) else void );
}
```

Benefits

- **Flexible binary interfaces**

  The Cake compiler uses DWARF debugging information to understand binaries. Like-named interface elements are related by default, bridging minor mismatches "for free".

- **Expressive pattern-matching**

  Pattern-matching syntax defines more complex correspondences between function calls or values either side of an interface mismatch.

- **Modular, maintainable glue logic**

  Cake's black-box view of binaries makes code resilient to internal changes within modules. Although strictly less powerful than invasive white-box approaches, we believe it to be practical for a wide range of composition tasks.

UNIVERSITY OF CAMBRIDGE

Stephen.Kell@cl.cam.ac.uk
http://www.cl.cam.ac.uk/~srk31