Component adaptation and assembly using interface relations

Or: a tool called Cake

Stephen Kell
Stephen.Kell@cl.cam.ac.uk

Computer Laboratory

University of Cambridge
There’s something about software…

Software is expensive and *inflexible*.

Tools assume:

- ground-up
- perfect fit
- don’t change
- never replaced

Reality: none of the above!
These programming tasks arise:

- as software evolves
- as new user requirements emerge
- as software *ecosystem* evolves

* e.g. alternative components become available
Cake is a tool for tasks like these. It is:

- a rule-based language
- ... for describing adapters
- declarative
- black-box
- convenient

Key contributions:

- expressive enough for realistic tasks
- ... context-sensitive, many-to-many relations
- ... evaluated on real tasks
Common approaches

- **edit or patch**
  - ![Diagram showing edit or patch approach]

- **glue coding**
  - ![Diagram showing glue coding approach]

- **abstraction layer**
  - ![Diagram showing abstraction layer approach]

Component adaptation... Cake – p.5/18
**Overview**

```
exists elf_reloc ("A.o") A;  // assume that object files...
exists elf_reloc ("B.o") B;  // ... contain debug info
derive elf_reloc ("whole.o") = link[A, B]
{
  /* your rules here! */
};
```
Simple adaptations

foo(...) ←→ bar(...);

Component adaptation... Cake – p.7/18
Simple adaptations

foo (...) $\leftrightarrow$ bar (...);

baz(a, b) $\leftrightarrow$ baz(b, a);
Simple adaptations

A

foo()  bar()

baz(a,b)  baz(b,a)

xyz(a)  xyz(a,42)

B

foo (...)  ↔  bar (...);
baz(a, b)  ↔  baz(b, a);
xyz(a)  ↔  xyz(a, 42);
Simple adaptations

foo (...) ←→ bar (...);

baz(a, b) ←→ baz(b, a);

xyz(a) ←→ xyz(a, 42);

values Br ←→ Tr
The Cake compiler generates wrapper functions from rules.
More complex adaptations

Real interfaces correspond less simply:

- non-1-to-1 mappings
- context-sensitive
- data, not just code
Many-to-many mappings

- mpeg2_dec_s
- mpeg2_sequence_t
  - width
  - height
  - display_width
  - display_height
  - chroma_width
  - chroma_height
- mpeg2_info_t
  - display_fbuf
  - sequence
- mpeg2_fbuf_t
  - buf[3]
  - id
- AVStream
  - codec
  - codec_type
  - codec_id
  - width
  - height
- AVCodecContext
  - size
  - AVPacket
  - size
- AVFrame
  - linesize[4]
  - data[4]

Component adaptation... Cake – p.9/18
Many-to-many mappings

values (dec: mpeg2_dec_s, info: mpeg2_info_s, seq: mpeg2_sequence_s, fbuf: mpeg2_fbuf_s) ←→ (ctxt : AVCodecContext, vid_idx: int, frame: AVFrame, p: AVPacket, s: AVStream, codec: AVCodec)

{ fbuf ←→ frame
  { buf[0] as line[seq.height] ptr ←→ data[0] as line[ctxt.height] ptr; }
} /* ←− more rules go here... */

This creates an association at run time (like a join table).
Context-sensitive mappings

Trace of start-up calls appropriate for the two libraries:

- `mpeg2_init()` \(\mapsto\) `dec`
- `fopen("...", "rb")` \(\mapsto\) `f`
- `mpeg2_info(dec)` \(\mapsto\) `info`
- `mpeg2_parse(dec)` \(\mapsto\) `STATE_BUFFER`
- `avcodec_init()` \(\mapsto\) `()`
- `av_register_all()` \(\mapsto\) `()`
- `av_open_file(...)` \(\mapsto\) `avf`
- `av_find_stream_info(avf)` \(\mapsto\) `()`
- `avcodec_find_decoder(...)` \(\mapsto\) `codec`
- `av_codec_open(...)` \(\mapsto\) `()`

Problem!

- One `info` call wants decoder object; other wants file.

Solution: context predication with name binding.

```ocaml
let dec = mpeg2_init(), ..., 
let f = fopen(fname, "rb"), ..., 
mpeg2_info(dec) \(\mapsto\) { /* now both f and dec are available */ }; 
```
“Passing objects” often means passing object graphs.

The Cake runtime traverses object structures automatically.

```c
values list_node_t <-> ListNode
{   data <-> item; }
values point_t <-> XYPoint;
```

Incidentally, note the following:

- name-matching is Cake’s default policy
- can insert stub code for value transformation (not shown)
Implementation

Compiler:

■ accepts Cake source file
■ emits wrapper functions as C++ code
■ consumes DWARF debugging information

Runtime:

■ allocator instrumentation
■ dynamic points-to analysis
■ “split heap” management, association tracking, . . .

Status: compiler still lagging language design, but WIP...
Compare Cake implementations with pre-existing adapters:

1. **p2k**: a filesystem adapter from the NetBSD OS
2. **ephy-webkit**: abstraction layer from Epiphany browser
3. **XCL** (subset of): compatibility layer for XCB X11 library

Summary outcome:

- **less code** (code written; various syntactic measures)
  - **p2k**: approx 70% reduction
  - **ephy-webkit**: approx 65% reduction
  - **XCL**: approx 30% reduction
- **less scattering of concerns**
int seek(struct puffs_usermount *pu, 
    puffs_cookie_t opc, off_t oldoff, 
    off_t newoff, struct puffs_cred *pcr)
{
    kauth_cred_t cred; int rv;

    cred = cred_create (pcr);
    VLE(opc);

    rv = RUMP_VOP_SEEK(
        opc, oldoff, newoff, cred);

    VUL(opc);
    cred_destroy (cred);

    return rv;
}

int remove(struct puffs_usermount *pu, 
    puffs_cookie_t opc, puffs_cookie_t targ, 
    struct puffs_cn *pcn)
{
    struct componentname *cn; int rv;

    cn = makecn(pcn);
    VLE(opc);
    rump_vp_incref (opc);
    VLE(targ);
    rump_vp_incref (targ);

    rv = RUMP_VOP_REMOVE(
        opc, targ, cn);

    AUL(opc);
    AUL(targ);
    freecn (cn, 0);
    return rv;
}
// rules concerning functions
p2k_node_seek(_, vn, oldoff, newoff, cred) → RUMP_VOP_SEEK(vn, oldoff, newoff, cred);

p2k_node_remove(_, vn as vnode_bump, tgtvn as vnode_bump, cn) → RUMP_VOP_REMOVE(vn, tgtvn, cn);

// rules concerning values
values puffs_cookie_t → ({VLE(that); that}) vnode;
values puffs_cookie_t ← ({VUL(that); that}) vnode;
values vnode_bump → ({VLE(that); rump_vp_incref(that);
  that}) vnode; // also bump refcount
values vnode_bump ← vnode; // unlock not required
values puffs_crd (cred_create(this)) → kauth_crd;
values puffs_crd ← (cred_destroy(this)) kauth_crd;
values puffs_cn (makecn(this)) → component_name;
values puffs_cn ← (freecn(this, 0)) component_name;

+ these rules contribute to other wrapper functions (28 total)
Similar tools with narrow domains or less expressiveness:

- Nimble (Purtilo, 1990), BCA (Keller, 1998)
- Jigsaw (Bracha, 1993), Knit (Reid, 2000)
- Swig (Beazley, 1996)
- C++ concept maps (Jarvi, 2007)
- Twinning (Nita, 2010)

Work focused on formalisation rather than implementation:

- Yellin & Strom, 1994; Bracciali, 2003
- subject-oriented composition (Ossher, 1995)

Clean-slate approaches to similar problems:

- Flexible Packaging (Deline, 2001)
Future work, conclusions, …

So far, Cake is

- a simpler way of writing short modular adapters
- a convenient tool for binary composition
- a step towards more compositional development

Cake is a platform for lots of potential future work.

- styles (for abstracting heterogeneous object code)
- white-box complement
- improved bidirectionality
- semi-automatic generation of Cake rules
Things I didn’t have time to mention

More language features:

- input versus output parameters
- error discovery & handling
- design for heterogeneity
- stub language (algorithms, lambdas, . . .)
- annotations, memory management adaptations, . . .

Repositories:

- http://www.cl.cam.ac.uk/~srk31/cake/

Thanks for your attention. Questions?
How Cake wins

- separate treatment of values from treatment of functions
- separate general from special cases
- name-matching
- black-box, binary, language-independent
- designed to accommodate heterogeneity
- make previously edit-requiring tasks black-boxable
- generates sequence recognition code automatically
- maintains object mappings (co-object relation) at runtime
- transitive treatment of object structures
- potential for bidirectional rules
```
switch binary

... gtk_entry_set_text(e, "...");
...

library call
directed into
wrapper

interposed code

co = ensure_co_object(e, REP_12);
rep_sync(e, co);
gtk_entry_set_text(co, t);

co = get_co_object(e, REP_12);
rep_sync(co, e);

libgtk20 binary

call library function
with alternate-rep
"co-objects"

void gtk_entry_set_text(e, t) {
    e->text = g_strdup(t);
    // ...
    return;
}

switch-accessible heap

libgtk20-accessible heap

...
Component adaptation... Cake – p.21/18

### p2k

<table>
<thead>
<tr>
<th>LoC (nb nc)</th>
<th>C</th>
<th>adjusted</th>
<th>Cake</th>
<th>remaining C</th>
</tr>
</thead>
<tbody>
<tr>
<td>605</td>
<td>523</td>
<td>133</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>3469</td>
<td>3137</td>
<td>1131</td>
<td>347</td>
<td></td>
</tr>
<tr>
<td>358</td>
<td>277</td>
<td>69</td>
<td>33</td>
<td></td>
</tr>
</tbody>
</table>

- wins: rule localisation (hugely), allocation

### ephy

<table>
<thead>
<tr>
<th>LoC (nb nc)</th>
<th>C</th>
<th>adjusted</th>
<th>Cake</th>
<th>remaining C</th>
</tr>
</thead>
<tbody>
<tr>
<td>525</td>
<td>513</td>
<td>161</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2529</td>
<td>2455</td>
<td>784</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>163</td>
<td>70</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- wins: rule localisation, many-to-many, graph exploration, pattern-matching
### XCL

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>adjusted</th>
<th>Cake</th>
<th>remaining C</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoC (nb nc)</td>
<td>380</td>
<td>315</td>
<td>189</td>
<td>42</td>
</tr>
<tr>
<td>tokens</td>
<td>2581</td>
<td>2328</td>
<td>1543</td>
<td>232</td>
</tr>
<tr>
<td>semicolons</td>
<td>187</td>
<td>148</td>
<td>107</td>
<td>19</td>
</tr>
</tbody>
</table>

- **problems**: abstraction gap, cross-rule commonality, more data types, more special cases, smaller subset of code (increasing returns?)