System support for adaptation and composition of applications

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

Computer Laboratory
University of Cambridge
First, a video

Note: not intended as a product endorsement!

Question: why doesn’t any old software Just Work like this?
Getting application code talking

It’s hard to get arbitrary bits of application code talking:

- different API calls
- different data structures
- different protocols
- different control structures, threading models...
- different language-level implementation details
  - storage management
  - threading semantics, concurrency primitives
- maybe no visible programmatic interface at all...

“Solutions”: standards; hand-written glue; don’t do it.
Mismatch is inevitable—must do adaptation.
Why the operating system is responsible

A problem just for the app developers? Not necessarily:

- OS has “component model”, controls communication
  - linkage / shared mem, pipes, sockets, filesystem...

- process/library notions are already too crude:
  - too big for precise isolation, security, accounting...
  - fail to capture dynamic structure below library level
    - witness research into “browser OSes”
  - would like to tackle the problem “closer to root”...
    - ...meaning in the OS interfaces
  - let’s rethink linker, dyn. loader, IPC interfaces
  - simplicity, coverage, adoptability, performance...

- *black-box* adaptation → *interposing* on communication
Adaptation, configuration languages

Glue code is tedious, fragile, error-prone.
- e.g. regex-based string rewriting
- e.g. implicit binding by name-matching

Want domain for specifying what and how of composition
- separate from programming languages
- explicitly captures wiring
- specialised towards adaptation

Goal: separation of functionality from integration details
- mix-and-match communication mechanisms
  - sockets, fs, subprocess, linkage, signals,…
- mix-and-match data encodings, protocols…
Concrete example – diagram

- calendar
- DBMS
- ODBC
- filesystem
- ad-hoc adaptation logic
- xml2sql
- sql2xml
- heap
Concrete example – code

```java
compound fsDbAdapter {
    exports [ fs_srv {
        int open(char list, int); int read(int, byte addr, int);
        int write(int, byte addr, int); void close(int); } ];
    imports [ db_client {
        char list list run_query(char list) } ];
    link obj_elf("C", "fs_db_adapter.o") [ xml2sql, sql2xml, realfs, heap ]
    { calfile = "myCalendar.xml"; rd_qstr = "SELECT * FROM Calendar";
      wt_qstr = "INSERT INTO Calendar VALUES ";
      fs_srv <- realfs {
        val fd = realfs.open("/ dev/null ", O_RDONLY);
        val inbuf = heap.newbuf(sql2xml(run_query(rd_qstr)))
        val outbuf = heap.newbuf(0)
        open(calfile, any mode) <- { return fd }
        read(fd, any dest, any len) <- { return heap.copy(inbuf, dest, len) }
        write(fd, any buf, any len) <- { return outbuf.append(buf, len) }
        close(fd) <- { run_query(wt_qstr + xml2sql(outbuf)) }
      }
    }
```

...adaptation and composition... – p. 7
What’s cool

Sounds straightforward. What’s cool?

- many “packagings”, same abstract model
  - data representation adaptations done implicitly
- embed language into dynamic loader
  - `dlopen()` arbitrary linkage units
  - `{library, file/socket}` handles unified as object handles
- simple in-OS object model helps language runtimes:
  - interoperability, e.g. multi-language shared heap
  - performance, e.g. avoid GC↔paging interaction
- extensible set of binary adaptation primitives
  - argument remapping; protocol adapter synthesis.
  - extend static data structures, initialisation
  - …

…adaptation and composition…. — p. 8
Status and current work

Current work extending Knit to support...

- ...simple compositions
- couple of different languages/packagings
- e.g. share Firefox’s history log with file manager

Blocked by current sideline: software visualisation

- want to identify candidate re-usable code...
- ...by inspection – need a picture e.g. linkage graph
- too many edges! aggregate them by cluster...

Thanks for listening. Questions?
Linkage graph… (rox-filer)

Wanted: decomposed representation with fewer edges
ROX filer after some ad-hoc clustering

After four rounds of head-scratching, it looks a bit better.

This was done mostly by deleting “pervasively-connected” nodes, together with their edges.
Why it’s not just...

- done by libraries?
  - need agreed-upon interfaces
- done by plug-in systems?
  - must conform to standard; not portable
- done by IDL compilers?
  - too specialized; fixed at compile-time
- done by component middleware?
  - little API adaptation; requires homogeneity
- done by aspect-oriented programming?
  - aspects bundle logic with interposition points
  - tend to therefore be codebase-specific
- done by web services etc.?
  - again, no API-level adaptation;
unit calendar {
    imports [ fs_srv { int open(char list, int); int read(int, byte addr, int); int write(int, byte addr, int); void close(int); }
    exports [ /* ... */ ];
    files { exec_elf("C", calendar) }
}

unit dbms {
    exports [ db_client { runQuery(char list) } ];
    files { obj_elf("C", odbc.o) }
}

unit xml2sql {
    exports [ xml2sql { xml2sql(char list) } ];
    files { exec_generic("execve_filter", xml2sql) }
}