SmPL: SimPLe SamPLes to Update Device Drivers

(supported by the ANR (FR) and the FTP (DK))

Gilles Muller   Yoann Padioleau   Julia Lawall

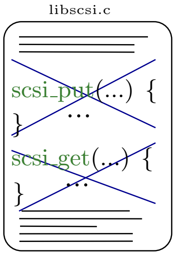
Ecole des Mines de Nantes
Gilles.Muller@emn.fr

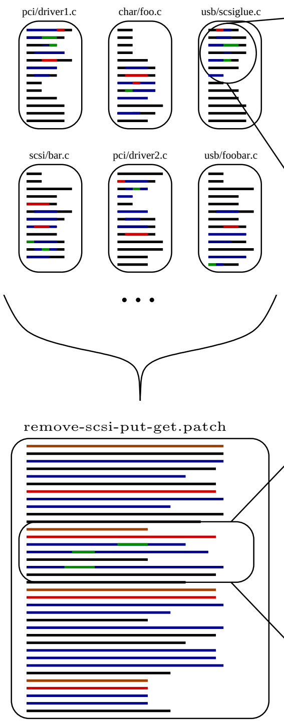
University of Copenhagen
julia@diku.dk

THE PROBLEM

Evolution in API of a generic library ⟹ Lots of Collateral Evolutions in clients of this library [Eurosys'06]

libscsi.c





Scsi_host *hostptr

```
static int usb_storage_proc_info(char *buffer,
    int length, int hostno, int inout) {
  struct us_data *us;
  char *pos = buffer;
  struct Scsi_host *hostptr;
  unsigned long f;

  if (inout) return length;
  hostptr = scsi_get(hostno);
  if(!hostptr) {
     return -ESRCH;
  }
  us = (struct us_data*)hostptr-hostdata0;
  if(!us) {
     scsi_put(hostptr);
     return -ESRCH;
  }
  pos++;
  scsi_put(hostptr);
  return pos+f;
 }
```

```
--- a/drivers/usb/scsiglue.c
+++ b/drivers/usb/scsiglue.c
@@ -164,33 +300,31 @@
- static int usb_storage_proc_info(char *buffer,
+ static int usb_storage_proc_info(Scsi_host *hostptr,
-     int length, int hostno, int inout) {
+     char *buffer, int length, int inout) {
  struct us_data *us;
  char *pos = buffer;
- struct Scsi_host *hostptr;
  unsigned long f;

  if (inout) return length;
- hostptr = scsi_get(hostno);
- if(!hostptr) {
-    return -ESRCH;
- }
 if(!us) {
-   scsi_put(hostptr);
    return -ESRCH;
 }
@@ -318,9 +342,6 @@
- scsi_put(hostptr);
  return pos+f;
}
```

Collateral evolutions are mostly done manually, because hard to script.

The program transformations require working on a high level representation of the program (syntactic and semantic, as in a compiler). Not just sed.

→ time consuming
(may involve 100 files, 1000 code sites)

→ error prone

Then, the modifications are transmitted to other Linux programmers via patch files.

Legend:
🟩 scsi get/put function calls to delete
🟦 dependent code to delete
🟥 code to add

OUR SOLUTION: A declarative easy-to-use transformation language to specify collateral evolutions [Eurosys'08].

Linux programmers exchange, read, and manipulate program modifications in terms of patches.
→ Our language is based around the idea and syntax of a patch, extending patches to Semantic Patches.

A single small Semantic Patch can modify hundreds of files, at thousands of code sites.

Semantic Patch Language (SmPL) by example

```
@@
struct SHT sht;
local function proc_info_func;
@@
    sht.proc_info = &proc_info_func;

@@
identifier hostptr, hostno, buffer, length, inout;
@@
  proc_info_func (
+     struct Scsi_Host *hostptr,
      char *buffer, int length,
-     int hostno,
      int inout) {
    ...
-   struct Scsi_Host *hostptr;
    ...
-   hostptr = scsi_get(hostno);
    ...
-   if(!hostptr) { ... }
    ...
-   scsi_put(hostptr);
    ...
  }
```

1. looks like real code, looks like a real patch
   A developer can construct a semantic patch by copy pasting existing driver code and then modifying and generalizing it to generate the semantic patch.
2. abstracts away differences in spacing, indentation, comments
3. abstracts away specific names given to variables and expresses constraints between code sites by declaring and using metavariables
4. declares arbitrary intervening code sequences, including straight-line code and arbitrary branching, with the '...' operator [POPL'09]
   Semantic patches work at the control-flow level.
5. abstracts away other variations using isomorphisms (e.g. `if(!hostptr)` ≡ `if(hostptr==NULL)`)

Features of SmPL that make semantic patches Generic to accomodate the many variations in device driver coding style.

→ Semantic patches developed for over 60 collateral evolutions

→ Over 180 Coccinelle-based patches integrated into Linux

http://www.emn.fr/x-info/coccinelle/