SmPL: A Domain-Specific Language for Specifying Collateral Evolutions in Linux Device Drivers

Yoann Padioleau [*]
Ecole des Mines de Nantes
Yoann.Padioleau@emn.fr

Julia L. Lawall
DIKU, University of Copenhagen
julia@diku.dk

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

Collateral evolutions are a pervasive problem in large-scale software development. Such evolutions occur when an evolution that affects the interface of a generic library entails modifications, *i.e.,* collateral evolutions, in all library clients. Performing these collateral evolutions requires identifying the affected files and modifying all of the code fragments in these files that in some way depend on the changed interface.

We have studied the collateral evolution problem in the context of Linux device drivers [1]. Collateral evolution is a significant problem in this context because device drivers make up over half of the Linux source code and are highly dependent on the kernel and driver support libraries for functions and data structures. Our EuroSys study [1] has shown that from one version of Linux to the next, collateral evolutions can account for up to 35% of the lines modified in such code. Moreover, those collateral evolutions may be complex, entailing substantial code reorganizations.

Currently, collateral evolutions in Linux are mostly done manually using a text editor, or with tools such as `sed`. The large number of Linux drivers, and complexity of the collateral evolutions, however, implies that these approaches are time-consuming and unreliable, leading to subtle errors when modifications are not done consistently. To address these problems, we propose a transformation language, SmPL,[1] to specify collateral evolutions. Because Linux programmers are used to exchange, read, and manipulate program modifications in terms of patches, we have built our language around the idea and syntax of a patch, extending patches to *semantic patches*.

Two important characteristics of semantic patches are:

- **Automation:** they can be applied automatically, and consistently. A single small semantic patch can modify hundreds of device drivers, at thousands of code sites.
- **Documentation:** they serve as a communication medium for all the actors involved in a collateral evolution because they document the collateral evolution formally and concisely.

We have already written 62 semantic patches. They cover the full taxonomy introduced in our EuroSys study.

[*] Contact author. No student authors.

[1] SmPL is the acronym for "Semantic Patch Language" and is pronounced "sample" in Danish, and "simple" in French.

A SmPL sample In Linux 2.5.71, collateral evolutions were performed in the "proc_info" functions of the SCSI drivers. An extract of a semantic patch specifying these collateral evolutions is as follows.

```
@@
struct SHT sht; local function proc_info_func;
identifier buffer, start, offset, length, inout, hostptr, hostno;
@@
  sht.proc_info = &proc_info_func;
  ...
  proc_info_func (
+      struct Scsi_Host *hostptr,
       char *buffer, char **start, off_t offset, int length,
-      int hostno,
       int inout) {
   ...
-  struct Scsi_Host *hostptr;
   ...
-  hostptr = scsi_host_hn_get(hostno);
   ...
-  if (!hostptr) { ... }
   ...
-  scsi_host_put(hostptr);
   ...
  }
```

The extract begins by declaring a set of metavariables between `@@`, that match any terms of the specified form. The rest of the extract specifies how to transform the driver code. As in a patch file, terms to remove and to add are indicated by `-` and `+`, respectively. The operator `...` indicates an arbitrary code sequence. The SmPL engine matches this specification to driver code modulo a set of isomorphisms; *e.g.* `if(!hostptr)` also matches code written as `if(hostptr==NULL)`. Furthermore, sequences are matched against the control-flow graph rather than the syntax tree. These features enhance the genericity of a semantic patch.

Is SmPL simple? A semantic patch describes a collateral evolution primarily in terms of ordinary C code. Thus a rule developer can often construct a semantic patch by copying and modifying existing driver code. Furthermore, a driver maintainer who wants to apply a semantic patch can easily understand its intent.

We are currently implementing the SmPL compiler. Work is also underway on using SmPL to specify the complete set of collateral evolutions required to update drivers from one version of Linux to a subsequent one.

[1] Y. Padioleau, J. L. Lawall, and G. Muller. Understanding collateral evolution in Linux device drivers. In *The first ACM SIGOPS EuroSys conference*, Leuven, Belgium, 2006.