Annex F

(informative)

Reflection API examples

F.1 Type information interface

The following is an example of a very simple use of the type information interface. It is the implementation of a method that receives a struct name as parameter, and prints out the fields with their types, and the methods with their parameter and return types.

```
print_struct(name: string) is {
   var s: rf_struct = rf_manager.get_type_by_name(name)
                      .as_a(rf_struct);
   outf("struct - %s\n",s.get_name());
   if s is a rf_like_struct (ls) {
      outf("\t inherits from - %s\n",
         ls.get_supertype().get_name()
         );
   };
   for each (f) in s.get_declared_fields() do {
      outf("\t field - %s: %s\n",
         f.get_name(),
         f.get_type().get_name()
         );
   };
   for each (m) in s.get_declared_methods() do {
      outf("\t method - %s()\n",m.get_name());
      for each (p) in m.get_parameters() do {
         outf("\t\t parameter - %s: %s\n",
            p.get_name(),
            p.get_type().get_name()
            );
      };
      if m.get_result_type() != NULL {
         outf("\t\t result type - %s\n",
            m.get_result_type().get_name()
            );
      };
   };
};
```

The following two files serve as a trivial design in order to show the output of the *print_struct* utility (the same code is referenced in the examples in other subclauses).

file1.e

```
type size_t: [big, medium, small];
struct packet {
   size: size_t;
   data: int (bits: 256);
   foo(id: int, name: string) is {
   };
};
```

```
extend sys {
    packets: list of packet;
    keep packets.size() > 3 and packets.size() < 7;
};
```

file2.e

```
import file1.e;
extend packet {
    corrupt: bool;
    foo(id: int, name: string) is also {
    };
    bar(): int is {
    };
};
```

This is output of running the utility on the code above:

```
Specman file2> print_struct("packet")
struct - packet
    inherits from - any_struct
    field - size: size_t
    field - data: int (bits: 256)
    field - corrupt: bool
    method - foo()
              parameter - id: int
              parameter - name: string
    method - bar()
              result type - int
```

F.2 Aspect information interface

The following code illustrates the way aspect information interface might be used. It is an implementation of a method that prints out the content of modules in terms of the type layers that they add to the overall design, a set of modules that are loaded (compiled) on top of Specman.

```
print_user_modules() is {
    for each (m) in rf_manager.get_all_user_modules() do {
        outf("module - %s\n",m.get_name());
        for each (tl) in m.get_type_layers() do {
            if tl is a rf_struct_layer (sl) {
                outf("struct layer - %s\n",
                    sl.get_defined_entity().get_name()
                    );
                for each (fd) in sl.get_field_declarations() do {
                    outf("\t\t field declaration - %s\n",
                        fd.get_defined_entity().get_name()
                        );
                };
                for each (ml) in sl.get_method_layers() do {
                    outf("\t\t method layer - %s (%s)\n",
                        ml.get_defined_entity().get_name(),
                        ml.as_a(rf_method_layer).get_method_kind()
                        .to_string()
                        );
                };
            };
        };
```

```
    };
};
```

Here is a possible output of this utility. In this case, it runs on the trivial design from the previous example (see F.1), namely modules `file1.e` and `file2.e`.

```
Specman file2> print_user_modules()
module - file1
struct layer - packet
                field declaration - size
                field declaration - data
                method layer - foo (is)
struct layer - sys
                field declaration - packets
module - file2
struct layer - packet
                field declaration - corrupt
                method layer - foo (also)
                method layer - bar (is)
```

F.3 Value query interface

It is hard to find an intuitive use for the object query interface which is simple enough to serve as an example. The following code is a very simple utility that prints out the state of objects recursively. It is somewhat artificial, since it prints out only enumerated and boolean fields.

```
print_struct_recursive(obj: any_struct) is {
   var s: rf_struct = rf_manager.get_struct_of_instance(obj);
   outf("instance of %s\n",s.get_name());
   for each (f) in s.get_fields() {
      if f.get_declaration().get_module().is_user_module() {
         outf("field %s - ",f.get_name());
         var vh: rf_value_holder = f.get_value(obj);
         if vh.get_type() is a rf_scalar (e) {
            outf("%s",vh.get_type().
            value_to_string(vh.get_value().unsafe()));
         } else if vh.get_type() is a rf_struct (s) {
            print_struct_recursive(vh.get_value().unsafe());
         } else if vh.get_type() is a rf_list (l) and
            l.get_element_type() is a rf_struct {
            outf("\n");
         var size: int = rf_manager.get_list_size(vh.get_value());
            for i from 0 to size-1 do {
               outf("%d: ",i);
               print_struct_recursive(rf_manager.
               get_list_element(vh,i).get_value().unsafe());
            };
         };
         outf("\n");
      };
   };
};
```

Here is the result of calling this method, given the little design of the previous examples (see F.1 modules `file1.e` and `file2.e`).

```
Specman file2> gen -seed = 5
Doing setup ...
Generating the test using seed 5...
Specman file2> print_struct_recursive(sys)
instance of sys
field packets -
0: instance of packet
field size - small
field data - -5319061515555392341
field corrupt - TRUE
1: instance of packet
field size - small
field data - 3230878320328792872
field corrupt - FALSE
2: instance of packet
field size - medium
field data - 2775930122720983980
field corrupt - TRUE
3: instance of packet
field size - big
field data - 2044827916054152830
field corrupt - FALSE
```