Libabigail: Application Binary Interface analysis using BTF, CTF and DWARF

Dodji Seketeli <dodji@redhat.com>

Claudiu Zissulescu <claudiu.zissulescu-ianculescu@oracle.com>

Linux Plumbers Conference, Vienna 18-20 September 2024

Outline of the talk

I. Introduction

II. Libabigail powers a set of tools

III. A re-usable library

IV. Supports multiple type information formats

V. Some timing information

VI. Questions / Discussions

Introduction

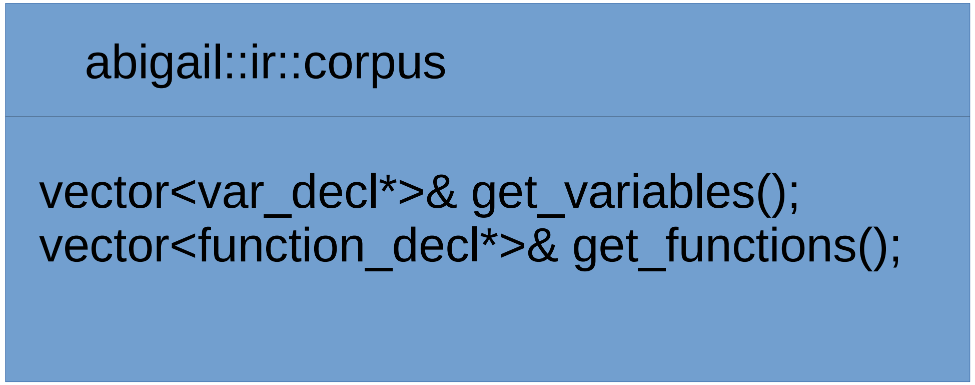
- Intended to:
  - represent artifacts of Application Binary Interfaces of shared libraries
    - Symbols, declarations, types
  - Compare these artifacts
    - Represent & analyze comparison results
  - Emit meaningful change reports
  - Operate from binaries (not source code)
- Started out by using
  - ELF (obviously)
  - DWARF
    - Ubiquitous for binaries generated in Fedora & RHEL ecosystem

A library powering a set of tools

- **ABIDIFF**
  - Compares exported declarations between two ELF binaries
  - Report about their ELF symbols changes
  - If binaries are accompanied by debug info then report about type changes
- **ABIPKGDIFF**
  - Compares exported declarations between binaries embedded in two packages
  - RPMs and Deb packages
    - Supports DWZ DWARF compression
  - Tarballs
- **ABIDW**
  - Emits textual representation (ABIXML) of the ABI of a binary
- **ABICOMPAT**
  - Test the ABI compatibility between an application an a shared library
- **KMIDIFF**
  - Compares the kernel/module interface between two Linux kernel trees
- **ABILINT**
  - Test if an ABIXML file can be loaded by the library
- **External tools**
  - **RPMINSPECT**
  - **check-uapi.sh**

A re-useable library

- Written in C++
- Around a central internal representation ...

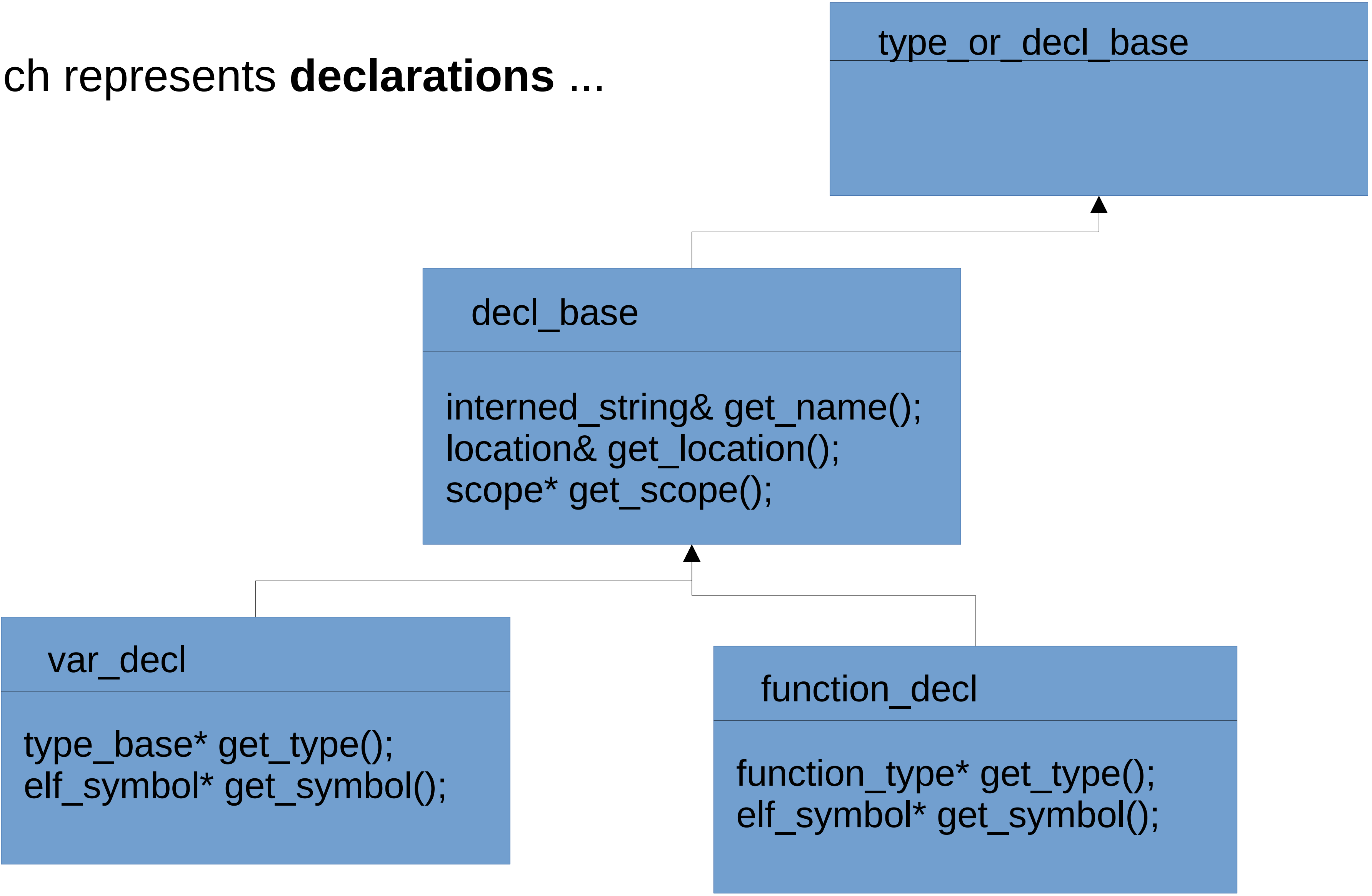


abigail::ir::corpus

```
vector<var_decl*>& get_variables();
vector<function_decl*>& get_functions();
```

A re-useable library

Which represents **declarations** ...



type_or_decl_base

decl_base

```
interned_string& get_name();
location& get_location();
scope* get_scope();
```

var_decl

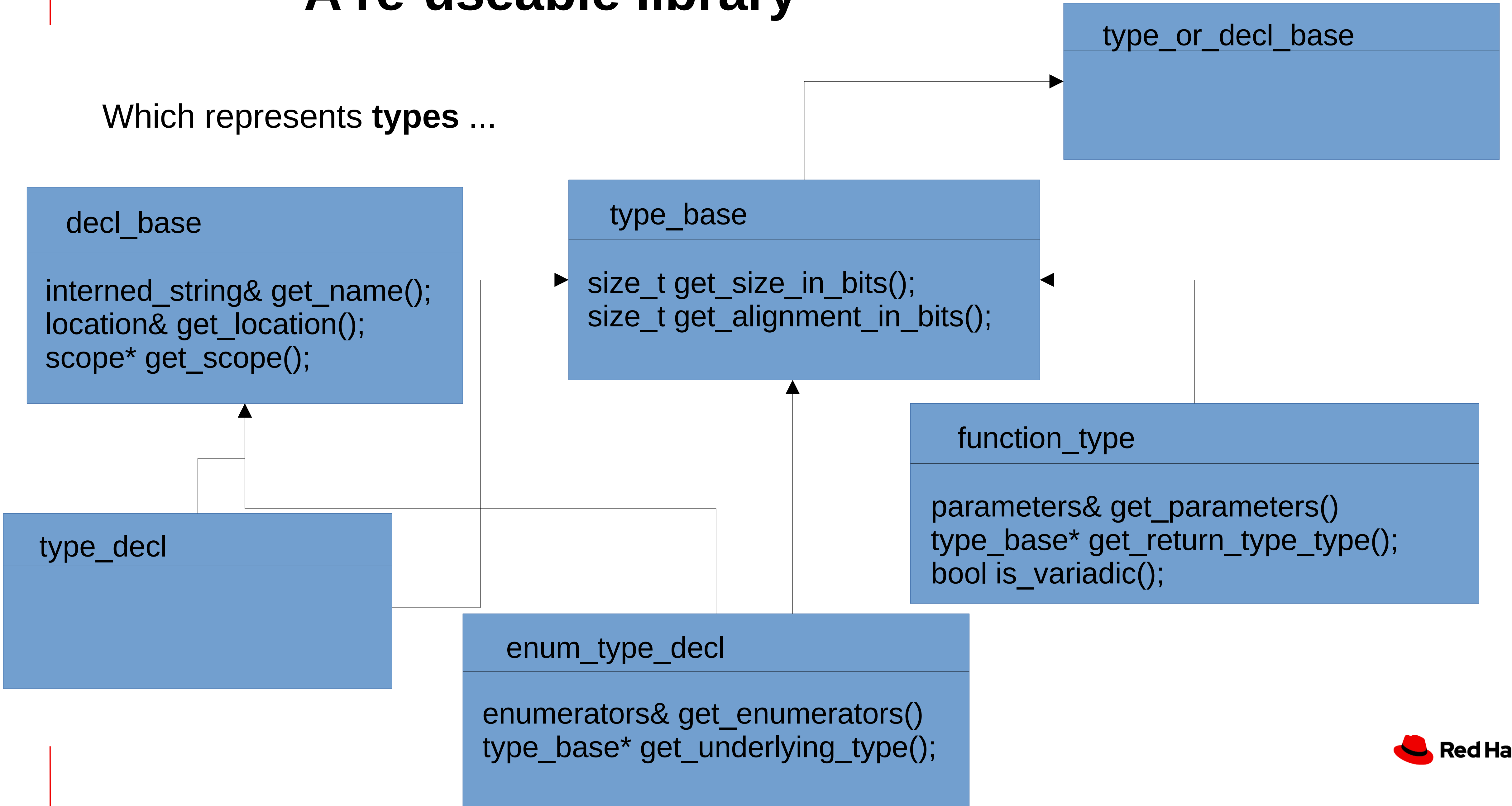
```
type_base* get_type();
elf_symbol* get_symbol();
```

function_decl

```
function_type* get_type();
elf_symbol* get_symbol();
```

A re-useable library

Which represents **types** ...



type_or_decl_base

decl_base

```
interned_string& get_name();
location& get_location();
scope* get_scope();
```

type_base

```
size_t get_size_in_bits();
size_t get_alignment_in_bits();
```

function_type

```
parameters& get_parameters()
type_base* get_return_type_type();
bool is_variadic();
```

type_decl

enum_type_decl

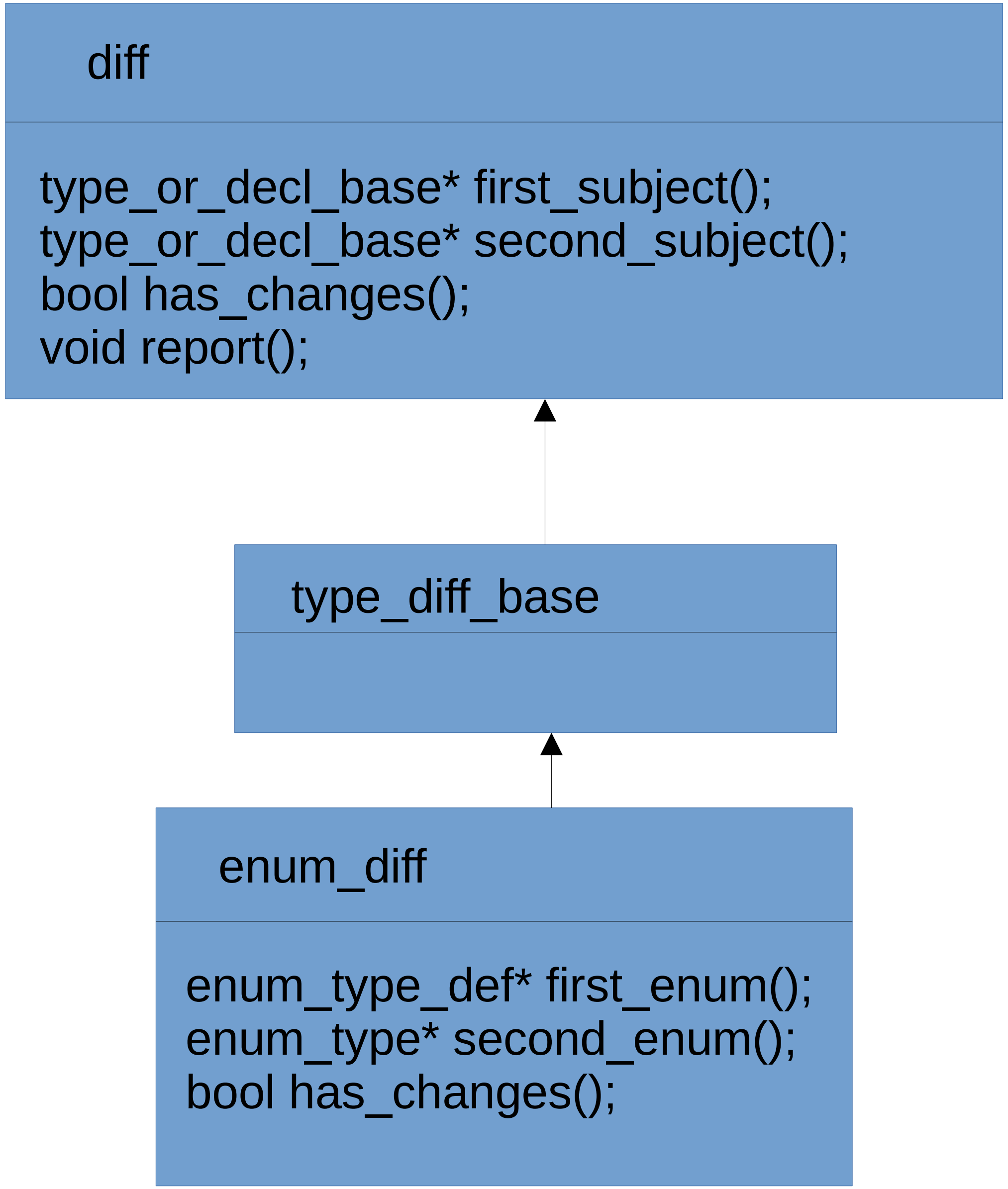
```
enumerators& get_enumerators()
type_base* get_underlying_type();
```

A re-useable library

Which represents **diffs** between ABI artifacts **...**

corpus_diff

```
corpus* first_corpus();
corpus* second_subject();
bool has_changes();
void report();
```



diff

```
type_or_decl_base* first_subject();
type_or_decl_base* second_subject();
bool has_changes();
void report();
```

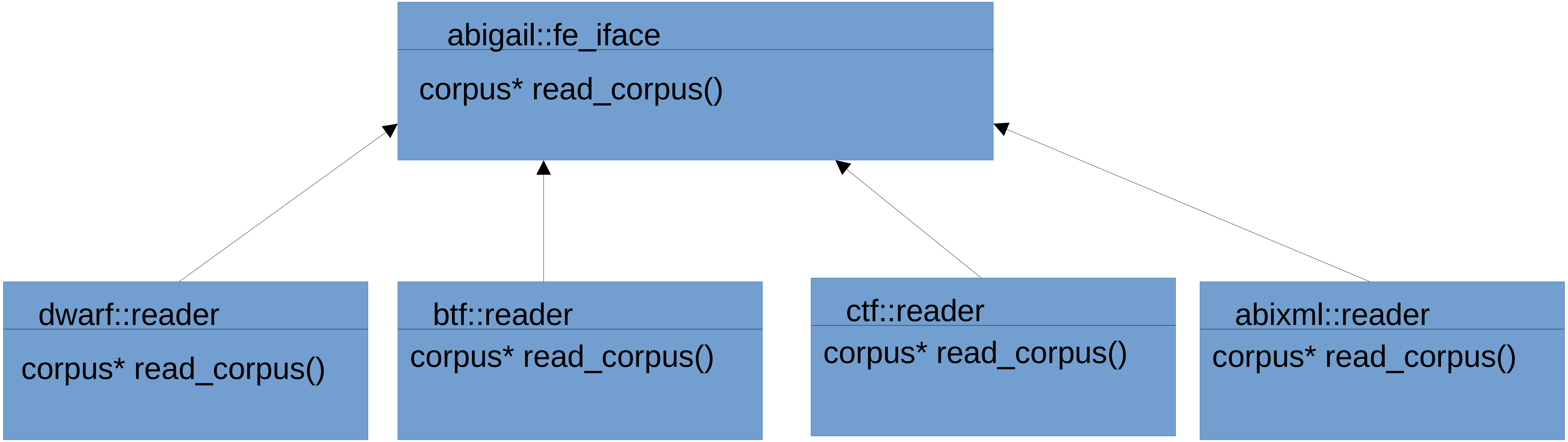
type_diff_base

enum_diff

```
enum_type_def* first_enum();
enum_type* second_enum();
bool has_changes();
```

Supports multiple type information formats

- An ABI corpus is created by the implementation of a **front-end** interface:



DWARF Front-end

- DWARF sports a high level of details
  - Support all languages
  - Every single translation unit of the binary is represented with:
    - All its types (and declarations).
  - Types defined in a header file are represented in all translation units that include it.
    - A given type T is likely to be represented (duplicated) in all translation units
- Need to **de-duplicate** all types
  - Across one binary
    - When analyzing one binary
      - abidw
  - Across two binaries
    - When comparing two binaries
      - abidiff
  - Across thousands binaries
    - When analyzing a Linux kernel and its modules
      - kmidiff
- Type de-duplication is done after creating the ABI corpus and before performing any comparison emitting an ABIXML.

BTF Front-end

- BTF available for the Linux kernel only
  - Just C and BPF
  - Much less information available than DWARF
    - E.g, no line information.
- Types are de-duplicated
  - Libabigail doesn’t have to perform type de-duplication after processing all
  - Processing is **much** faster and smaller than DWARF.

CTF Front-end

- CTF available for C programs
  - Emitted by GCC
  - Much less information available than DWARF
    - E.g, no line information.
- Types are de-duplicated
  - Libabigail doesn’t have to perform type de-duplication after processing all
  - Processing is **much** faster and smaller than DWARF.
  - Faster than BTF.

Some timing information

- Using the kernel at sourceware.org/git/libabigail-tests.git
  - Enterprise kernel with mode than 3000 modules
  - Using the kmidiff tool
- Using the CTF front-end
  - 30 seconds / 1,5GB of max resident memory size
- Using the BTF front-end
  - 1m:16 secs / 2GB of max resident memory size
- Using the DWARF front-end
  - 35 minutes / 8GB of max resident memory size

Discussions

- https://sourceware.org/libabigail
- https://sourceware.org/libabigail/apidoc/
- https://sourceware.org/libabigail/manual/
- irc://irc.oftc.net#libabigail

Thank you!