Bindings Interop

at BlinkOn 9, by yukishiino@ and peria@

Topics

Interop on Cross origin properties

Yuki Shiino (yukishiino@chromium.org)

Re-architect IDL compiler

Hitoshi Yoshida (peria@chromium.org)

Interop on Cross origin properties

Yuki Shiino (yukishiino@chromium.org)

Goal

Bindings team (TOK) is now actively working to support
HTML 7.2.3.1 CrossOriginProperties with conforming to the spec
as a long term goal.

Cross origin properties are properties accessible across *origins*, such as

- `window.postMessage()`
- `window.parent`
- `location.href` (assignment only)

Current spec/interop violation (1 of 2)

The current implementation of Blink is different from the spec.

Web IDL (and HTML) says:

IDL attributes must be ES accessor properties.
(i.e. get + set accessor ES functions)

Blink implementation is:

Cross origin IDL attributes are ES **data** properties (with C++ hook functions).
(Other IDL attributes are ES accessor properties.)

Current spec/interop violation (2 of 2)

The current implementation of Blink is different from the spec.

HTML 7.4.5 WindowProxy's [[GetOwnProperty]] says:

the same origin-domain ⇒ OrdinaryGetOwnProperty
otherwise ⇒ CrossOriginGetOwnPropertyHelper

Blink implementation is:

always CrossOriginGetOwnPropertyHelper

```
// in windowA (= realm A)
windowB.postMessage ⇒ «postMessageInRealmA»
```

Blocking issue: Incumbent realm

A: Okay, let's fix it.

B (or me): Well… we cannot fix it right away because Blink does ***NOT*** support incumbent realm. If we fixed it, the navigation gets broken. We first need to support incumbent realm.

A: What is the incumbent realm?

Q: What's the Incumbent realm?
A: Realm of "the **most-recently-entered** author function" (HTML 8.1.3.5)

```
// in windowA
function FuncA() { windowB.FuncB(); }
// in windowB
function FuncB() { windowC.location.href = "url"; }
windowA.FuncA();
```

When running `windowC.location.href`'s setter, the *most-recently-entered author function* is `FuncB`, i.e. the incumbent realm = realm B.

Navigation must be resolved relative to the incumbent's URL.
(HTML Location-object navigate)

```
// in windowA
function FuncA() { windowB.FuncB(); }
// in windowB
function FuncB() { windowC.location.href = "url"; }
windowA.FuncA();
```

When running `windowC.location.href`'s setter function,

the current realm = realm C   (in case of the same origin-domain)

the incumbent realm = realm B

In general, current != incumbent

Current impl in Blink w/o incumbent (attribute)

```
// in windowA
function FuncA() { windowB.FuncB(); }
// in windowB
function FuncB() { windowC.location.href = "url"; }
```

`href` is a ***data*** property (= no setter function)

⇒ when running C++ callback for `href`, the current realm = realm B,
that matches the incumbent realm by HTML.
(There is no ES function created in realm C because `href` is a ***data*** property.)
⇒ Blink is using the **current** realm instead of
the incumbent realm when navigating.

Current impl in Blink w/o incumbent (operation)

```
// in windowA
function FuncA() { windowB.FuncB(); }
// in windowB
function FuncB() { windowC.location.replace("url"); }
```

`replace` is a function created in realm B (even when the same origin-domain)

⇒ when running `replace`, the current realm = realm B, that matches the incumbent realm by HTML.

⇒ Blink is using the **current** realm instead of the incumbent realm when navigating.

Definition of the incumbent realm

Let's support the incumbent realm in order to fix cross origin properties.
The incumbent realm is the realm associated with either of

- the **most-recently-entered** author function, or
- the author function **originally scheduled a callback** (HTML 8.1.3.5)

The second definition is necessary because it's possible that there is no author function on the call stack.

Example of no author function on the call stack

```
function FuncC() {
  setTimeout(location.replace.bind("URL"));
}
```

Scheduled callback is `location.replace.bind("URL")`, which is not an author function because `bind` does not create a new function (`bind` creates a bound function exotic object).

In this case, the second definition is used.
`FuncC` scheduled the callback ⇒ the incumbent realm is realm C.

Important findings

IDL callback function **!=** ES function

IDL callback function == ES function

+ **incumbent at time of scheduling**

Most of people are confused about this point.

More or less, the existing callbacks in Blink are wrong.

Mission updated: Fix all the callbacks

Let's fix all the IDL callback functions + IDL callback interfaces in order to support the incumbent realm, that is necessary when fixing cross origin properties.

note: Web IDL only supports two kinds of callbacks:
callback function and callback interface.

The current status

We've done so far...

- Fully rewrote bindings support for IDL callback function/interface to be capable of handling the incumbent realm.
- Migrated 20 clients of callback functions to use the new ones.
- Migrated 12 clients of callback interfaces to use the new ones.

and we'll work on...

- Migrate NodeFitler and EventListener callback interfaces.
- Fix some more unique clients of callback:
  custom elements, CSS Painting API, etc.

Summary

Bindings team is working to support...

- cross origin properties (final goal)
- incumbent realm (needed for cross origin properties)
- fix all clients that use callbacks (needed for incumbent realm)

It's a long way to go... (a kind of yak-shaving ;)

IMPORTANT: callback != ES function

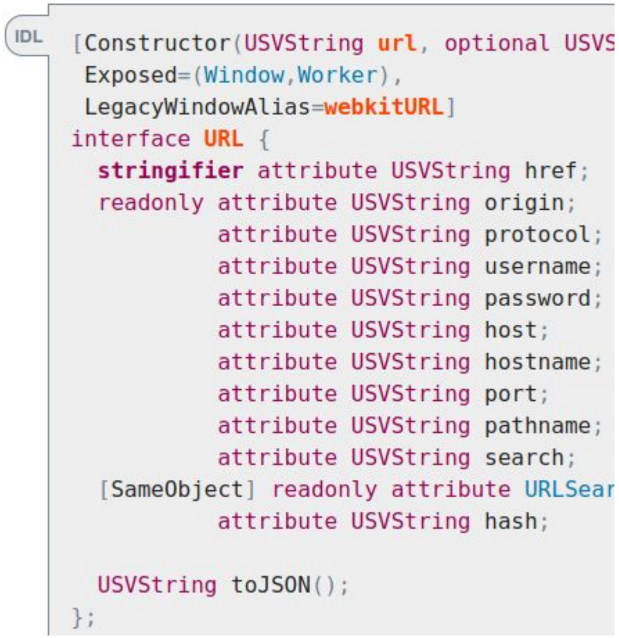
Unless it's an IDL callback, you must not invoke an arbitrary ES function.

Re-architect IDL compiler

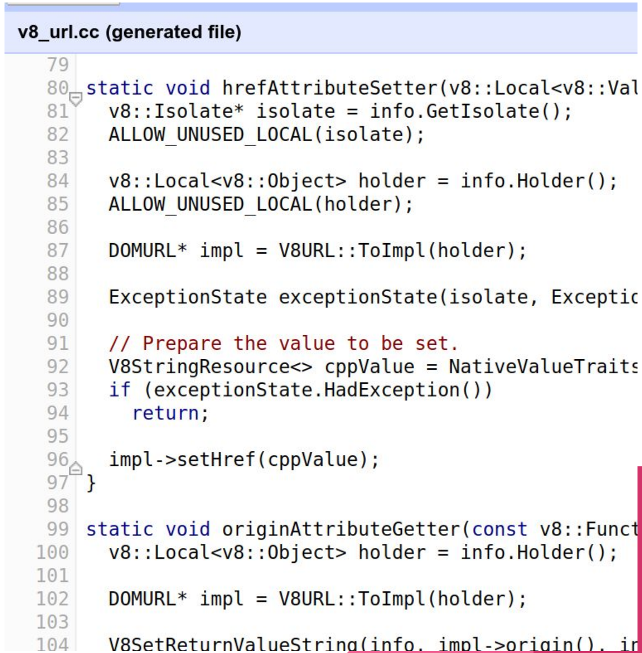
Hitoshi Yoshida (peria@chromium.org)

What is IDL compiler

IDL compiler converts Web APIs to C++ bindings code



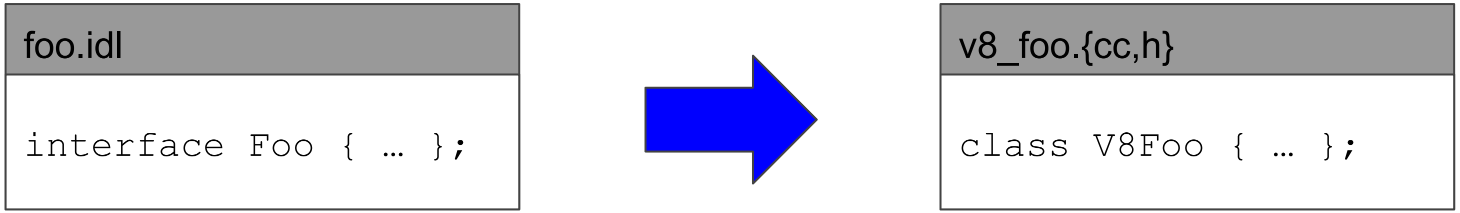
```
IDL
[Constructor(USVString url, optional USVS
 Exposed=(Window,Worker),
 LegacyWindowAlias=webkitURL]
interface URL {
  stringifier attribute USVString href;
  readonly attribute USVString origin;
           attribute USVString protocol;
           attribute USVString username;
           attribute USVString password;
           attribute USVString host;
           attribute USVString hostname;
           attribute USVString port;
           attribute USVString pathname;
           attribute USVString search;
  [SameObject] readonly attribute URLSear
           attribute USVString hash;

  USVString toJSON();
};
```



v8_url.cc (generated file)

```

static void hrefAttributeSetter(v8::Local<v8::Val
  v8::Isolate* isolate = info.GetIsolate();
  ALLOW_UNUSED_LOCAL(isolate);

  v8::Local<v8::Object> holder = info.Holder();
  ALLOW_UNUSED_LOCAL(holder);

  DOMURL* impl = V8URL::ToImpl(holder);

  ExceptionState exceptionState(isolate, Exceptic

  // Prepare the value to be set.
  V8StringResource<> cppValue = NativeValueTraits
  if (exceptionState.HadException())
    return;

  impl->setHref(cppValue);
}

static void originAttributeGetter(const v8::Funct
  v8::Local<v8::Object> holder = info.Holder();

  DOMURL* impl = V8URL::ToImpl(holder);

  V8SetReturnValueString(info, impl->origin(), ir
```

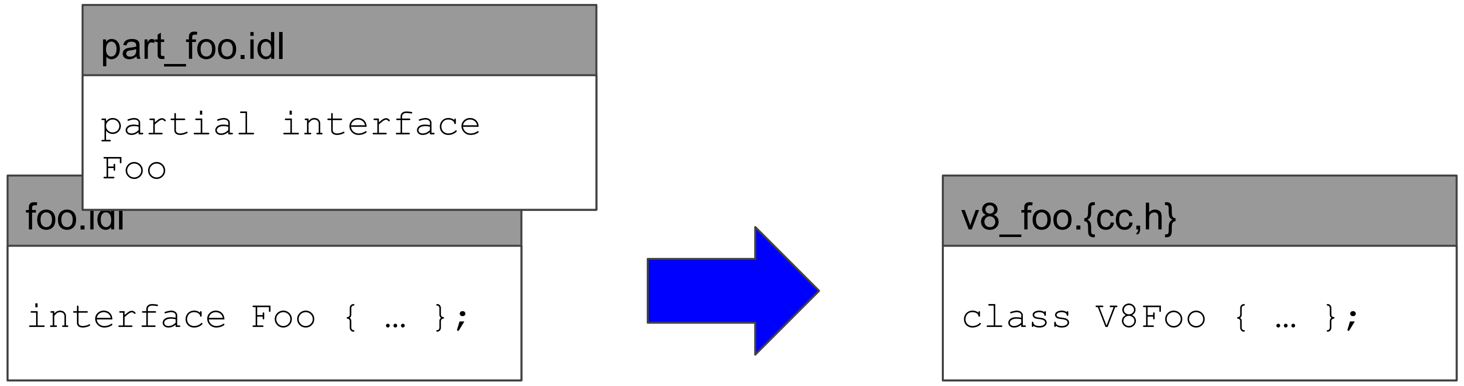
How does IDL compiler work?



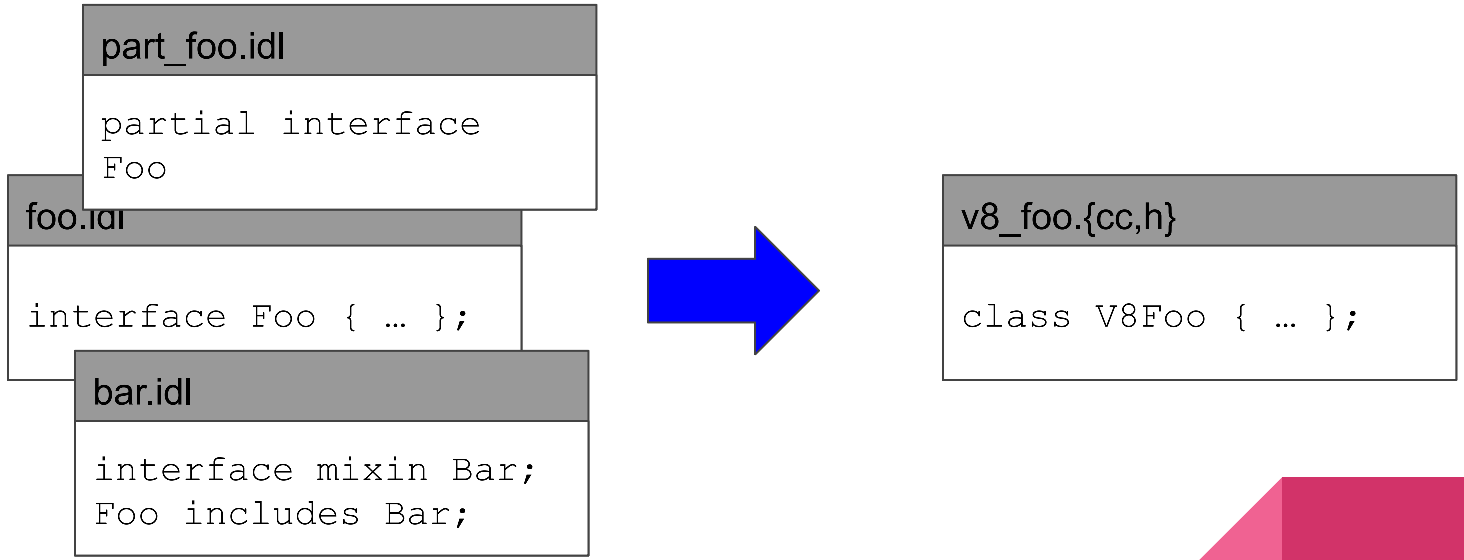
```
interface Foo { … };
```

```
class V8Foo { … };
```

How does IDL compiler work?



How does IDL compiler work?



part_foo.idl

```
partial interface
Foo
```

foo.idl

```
interface Foo { … };
```

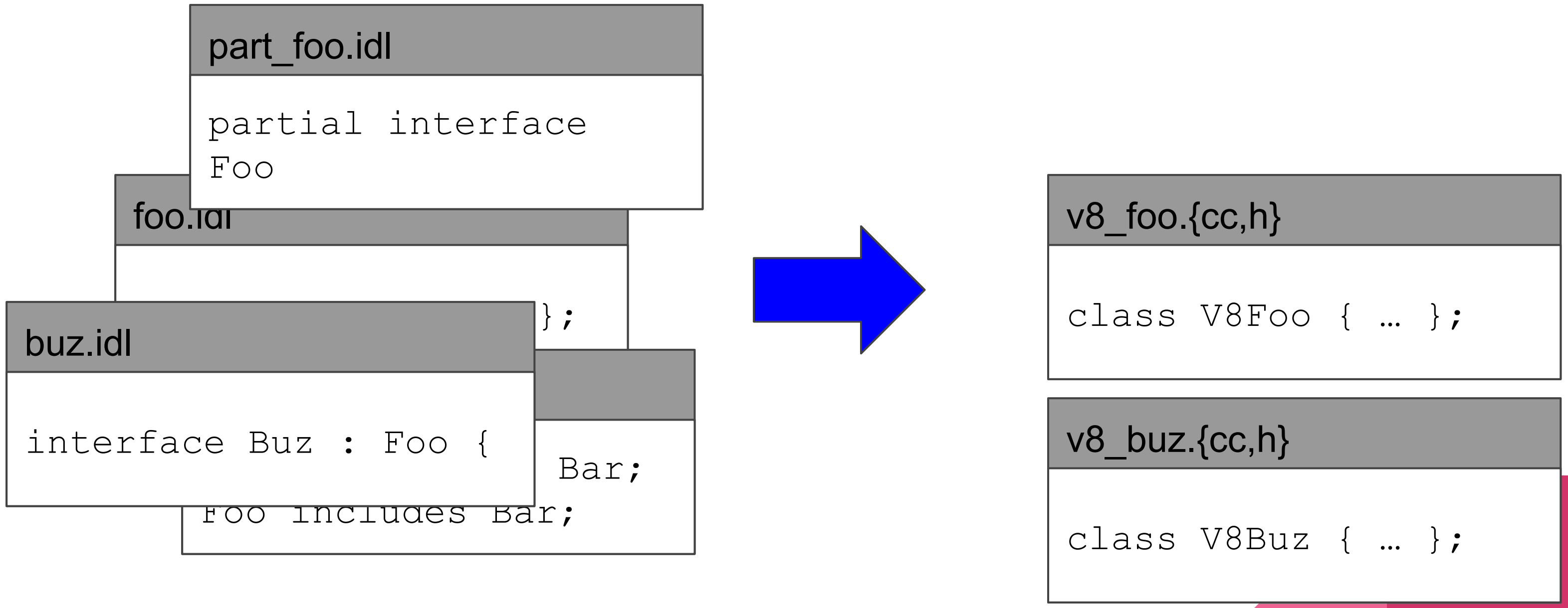
bar.idl

```
interface mixin Bar;
Foo includes Bar;
```

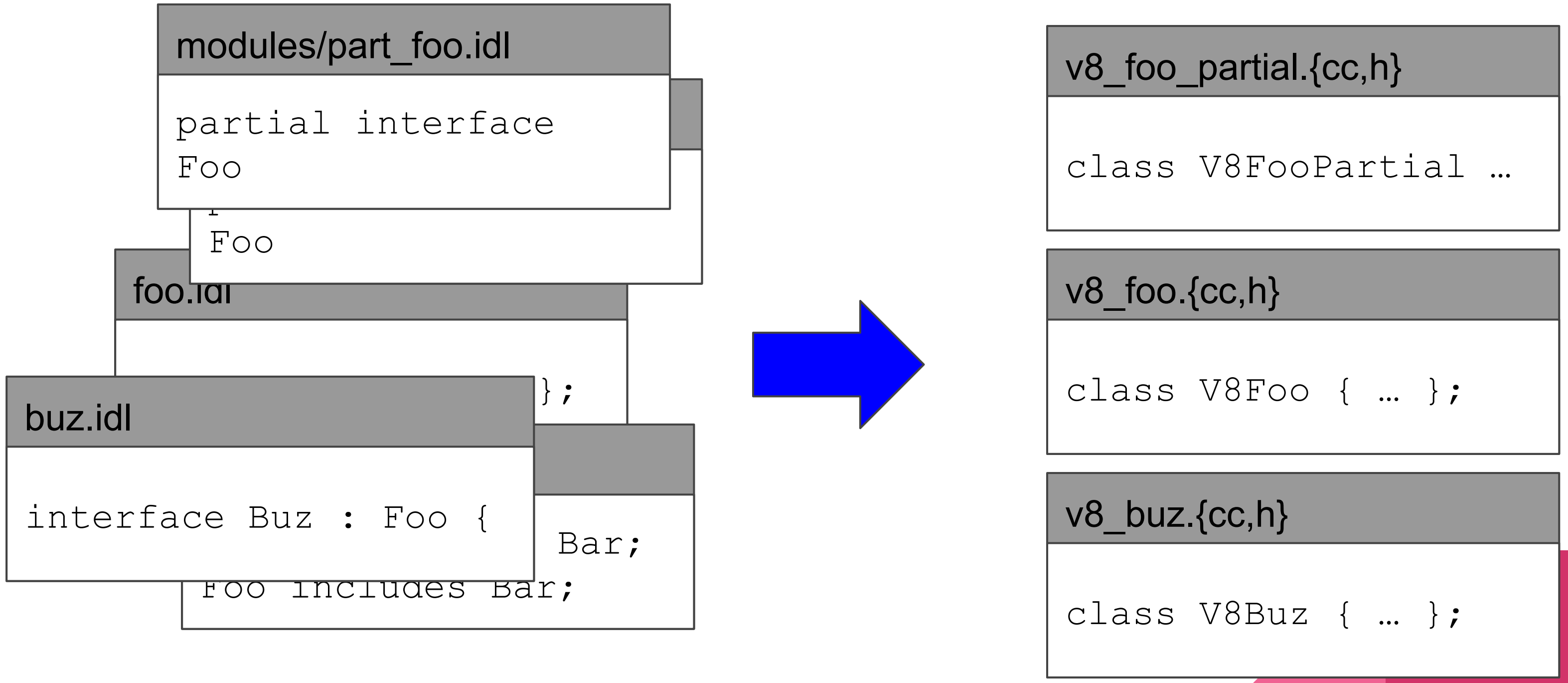
v8_foo.{cc,h}

```
class V8Foo { … };
```

How does IDL compiler work?



How does IDL compiler work?



Time flies

Since the current IDL compiler was designed in 2014, many Web specs, including the spec of WebIDL, are being updated.

And we had a big change in 2015; the componentization.

These changes introduced many issues around IDL compiler.

Issues around IDL compiler

Unexpected behaviors

- 809368: build fail when using PutForwards keyword in idl
- 752877: Ignores inheritance of anonymous setters and getters

Implementation limits

- 656517: Support partial interface for mix-in IDL
- 672978: Putting a subset of constructors behind a runtime enabled flag

Issues around IDL compiler

Support spec features

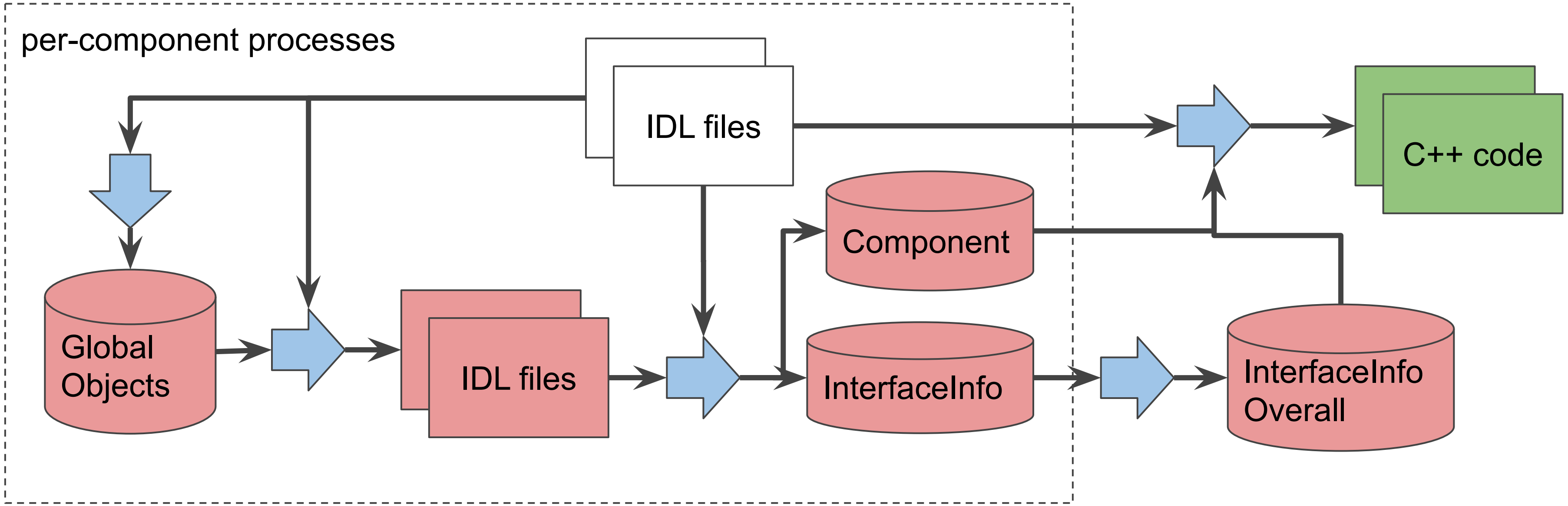
- 727971: Support "namespace" definition in WebIDL
- 781257: Add support for WebIDL mixins

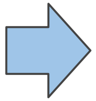
Roots of the pain

- Type references were limited
- Componentization made it complex

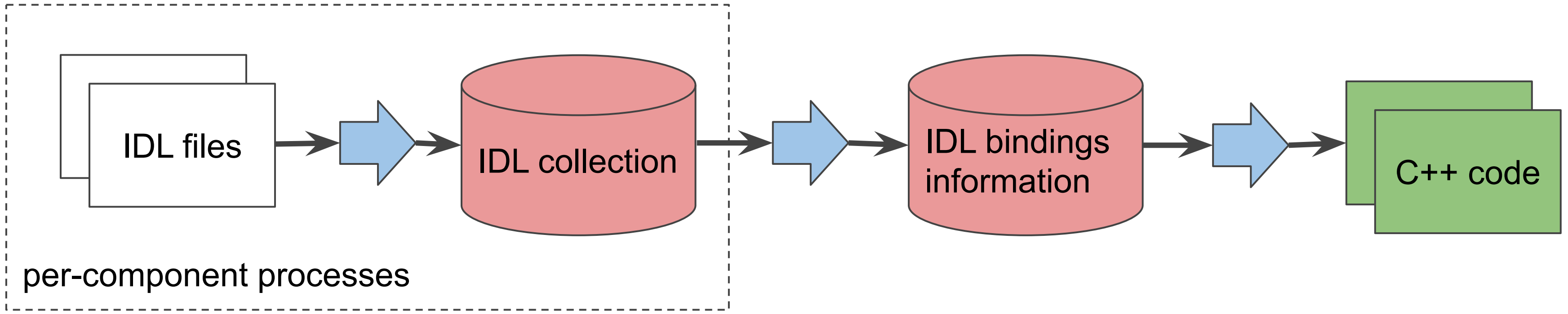
Re-architect

Current workflow of IDL compiler




… GN task

How the new IDL compiler will work



Key points of the new workflow

- Generate a unique global repository
  - It contains all information in .idl files.
  - Structured objects are linked directly.
- Change styles in C++ code
  - Chromium coding style.
  - How to install properties on templates and instances.
- Use Mako template library
  - We can write python style code inside templates.