Refinement.js

Yet another contract library for JavaScript

Jinwei Long (@NiceKingWei)

2018.6.10

xWIDL

- **Modular** and **Deep** *JavaScript* API Misuses Checking Based on eXtended WebIDL
- ACM SPLASH 16’ Student Research Competition
- Zhen Zhang(@izgzhen) , *University of Science and Technology of China*

xWIDL: Background

- JavaScript is a scripting language with a lot of platform APIs. (as part of browser, node.js runtime ...)
- JavaScript is also error prone, especially when we use platform APIs.
- Here is an example:

```
var blob = new Blob("Hello");
blob.close();
var url = URL.createObjectURL(blob);
```

xWIDL: Static Analyzers helps a lot

- TAJS: Type Analyzer for JavaScript
  - Abstract Interpretation
  - Detect null and undefined values, type errors and other potential bugs
- SAFE:
  - Modular Static Analysis Platform
- But they can’t do anything with the platform APIs. Though we can hardcode them inside the analyzer, but we should not.

xWIDL: webIDL and xWIDL

- WebIDL is an interface definition language for platform APIs such as DOM and WebGL

```
interface Blob {
    readonly attribute boolean isClosed;
    void close();
};
partial interface URL {
    static DOMString createObjectURL(Blob blob);
};
```

xWIDL: webIDL and xWIDL

- We can extend webIDL:

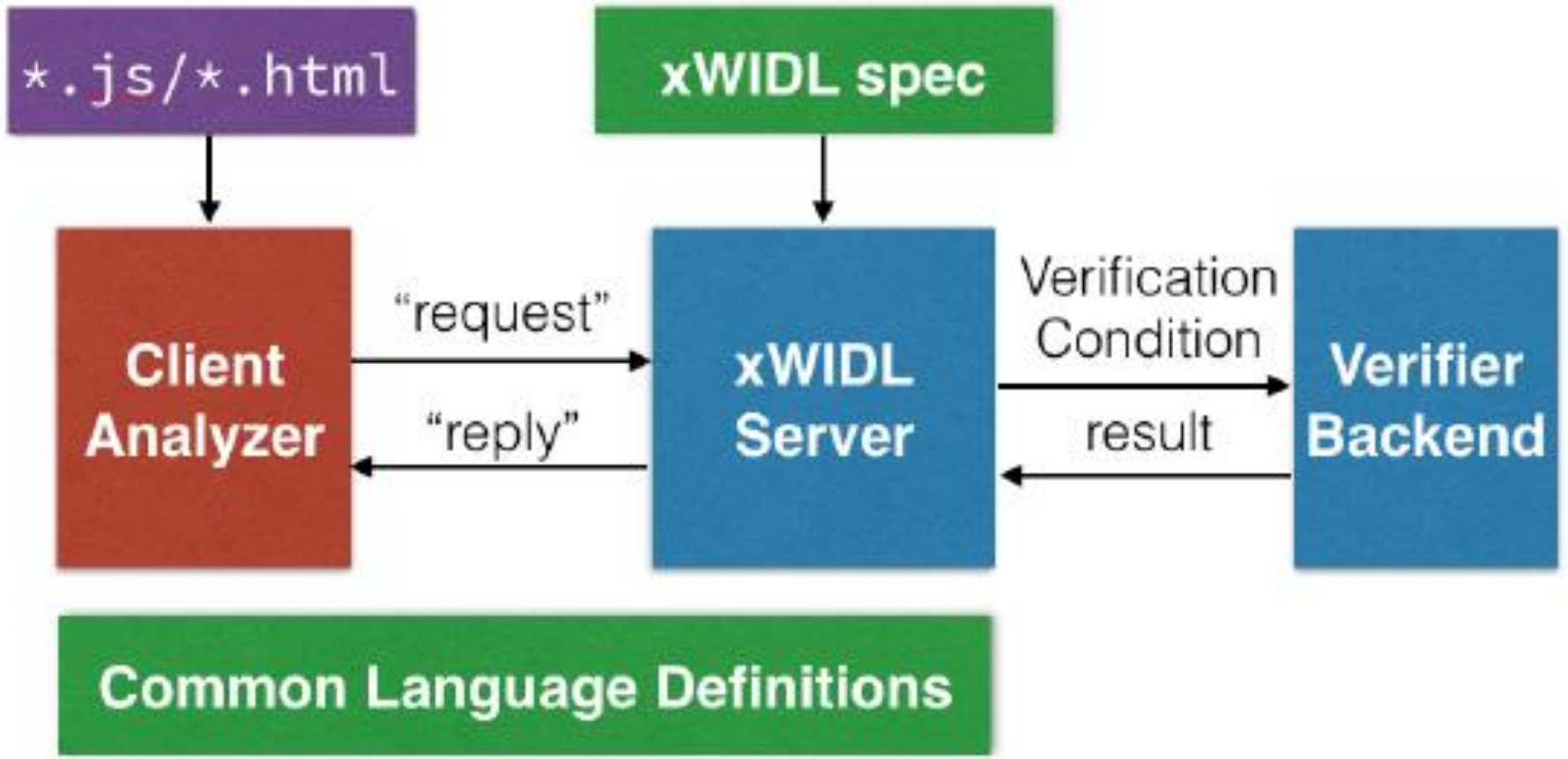
```
interface Blob {
        readonly attribute boolean isClosed;
        void close();
        ///- effects { isClosed := true; }
};
partial interface URL {
        static DOMString createObjectURL(Blob blob);
        ///- requires (blob.isClosed == false)
};
```

xWIDL: How to verify?

- Using Dafny


```
method DutchFlag(a: array<Color>)
  requires a ≠ null modifies a
» ensures ∀ i,j · 0 ≤ i < j < a.Length ⟹ Ordered(a[i], a[j])
  ensures multiset(a[..]) == old(multiset(a[..]))
{
  var r, w, b := 0, 0, a.Length;
» while w ≠ b
    invariant 0 ≤ r ≤ w ≤ b ≤ a.Length;
    invariant ∀ i · 0 ≤ i < r⟹a[i] == Red
    invariant multiset(a[..]) == old(multiset(a[..]))
  {   match a[w]
      case Red ⇒
        a[r], a[w] := a[w], a[r];
        r, w := r + 1, w + 1;
      case White ⇒
        w := w + 1;
      case Blue ⇒
        b := b - 1;
```

xWIDL: RPC Structures



xWIDL: Good Ideas

- Introduce specifications(or verification conditions) to program analysis. Because non-trivial constraints do exist in platform APIs (such as implicit registration)

```
var gl= document.getElementById("canvas").getContext("webgl");
var buf = gl.createBuffer();
/* WebGLBuffer object buf is registered in gl implicitly */
/* now we get a different WebGL context gl2 from somewhere else */
gl.bindBuffer(buf);  // Correct
gl2.bindBuffer(buf); // Incorrect
```

- Programmers don't like specifications, but want to get more information to locate their bugs

xWIDL: Weakness

- Dafny is too restrict to use. It needs so many specifications that the program can't generate automatically.

```
method m(n: nat)
{
        var i: int := 0;
        while i < n
                invariant 0 <= i <= n
        {
                i := i + 1;
        }
        assert i == n;
}
```

We must find a new way towards destination

- Related works
  - Gradual Refinement Types
    - Refinement Types: Type + Propositions
    - Gradual Typing: some values have static types and others have dynamic types
  - Design by Contract
    - Eiffel language: design by contract
    - Code Contract: static checking, dynamic checking and document generation
  - Program Analysis
    - Abstract Interpretation
    - Symbolic Execution

Exciting Experiments

- Non-trivial contracts can be transformed to trivial errors such as null pointer exception

```
function Blob(){
        this.ghost_available = function(x){return x;};
        this.close = function(){
                this.ghost_available = null;
        }
}
function createObjectURL(blob){}
var blob = new Blob();
blob.ghost_available(createObjectURL)(blob);
var blob_alias = blob;
blob_alias.close();
blob.ghost_available(createObjectURL)(blob);
```

Exciting Experiments

- We can perform this for all functions

```
if(precondition){
        var result = (function(){
                do_something();
                if(!assert_condition) undefined();
        })();
        if(!postcondition(result)) undefined();
} else {
        undefined();
}
```

Exciting Experiments

- Sounds great. But there are still some difficulties.
- Where can you get CFG(Control Flow Graph) ?
  - x.fun(): what's the variable x's type? where is the definition of fun?
  - function sqrt(x) requires x>=0, if we call sqrt(-1), the analyzer will tell us x>=0 will be violated but the won't tell us which call violates it.
- Introducing more functions means imprecise results

That’s what refinement.js does

- Transform specifications to normal JavaScript code
- Add some guide for analyzer to analysis the code with more accuracy
- Implemented in TypeScript
- Try it now! (Node.js and JRE are needed)

```
git clone https://github.com/NiceKingWei/refinement.js.git
cd refinement.js/
sudo npm install –g
rfjs examples/1.js
```

That’s what refinement.js does

- examples/1.js

```
1   function sqrt(x) {
2       requires(x>=0);
3       var result = Math.sqrt(x);
4       assert(result>0);
5       return result;
6       ensures(function(res){return res>=0;});
7   }
8
9   function fun(){}
10
11  var is_prime = function(n){
12      for(var i=2;i*i<n;i++) if(n%i==0) return true;
13      return false;
14  }
15
16  sqrt(4);
17  sqrt(-1);
18  sqrt(0);
19  is_prime(3);
```

That’s what refinement.js does

- examples/1.js

Result

```
examples/1.js:4:38: [definite] Assertion failed
examples/1.js:6:49: [definite] The postcondition might not hold
examples/1.js:12:14: [definite] The conditional expression is always false
examples/1.js:17:1: [maybe] The precondition might not hold
examples/1.js:18:1: [maybe] The precondition might not hold
```

Future Work

- More precise analysis and more friendly report
- Allow dynamic checking of specifications
- Deterministic/Abstract Abstract Interpreter (I will focus on this problem)

Thank you!