www.italiancpp.org

C++ and UIs an unorthodox approach

Daniele Pallastrelli

C++ Day 2017
2 Dicembre, Modena

C++ today

- Games
- Embedded
- OS, VM &c
- Simulations & HPC
- …

In general:

- High Performance / low latency applications
- Hardware interaction
- Legacy code

How do we usually write UIs for C++?

- Native APIs (Win API, Xlib, Cocoa)

- Class Frameworks (MFC, WTL, .NET, …)

- Widget cross-platform frameworks (FLTK, …)

- Huge cross-platform frameworks
  (Qt/copperspice, wxWidgets, GTK+, Ultimate++ , …)

- Modern C++ focused on UIs (Nana C++, …)

How do we usually write UIs for C++?

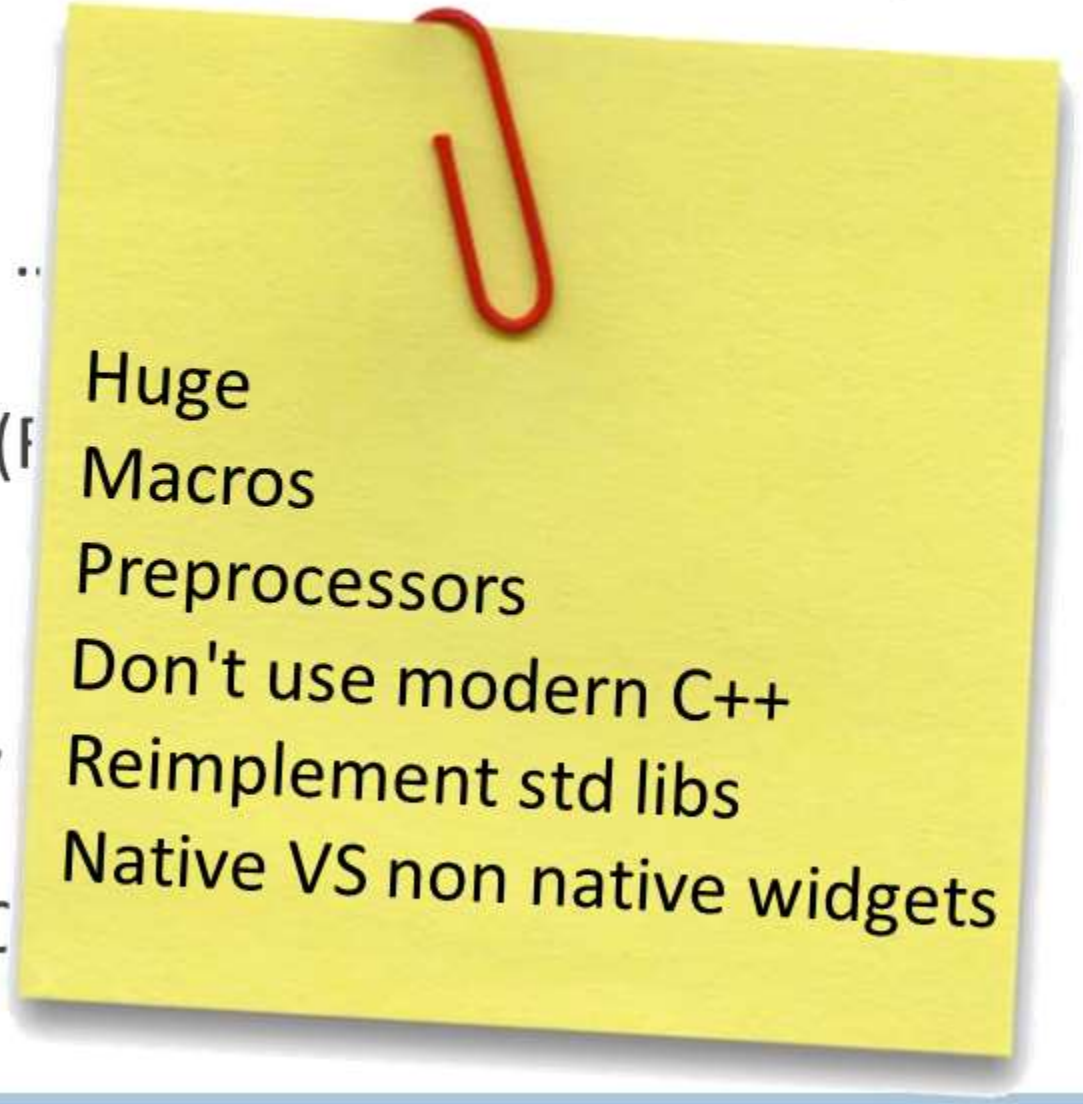
- Native APIs (Win API, Xlib, Cocoa)

- Class Frameworks (MFC, WTL, .NET, ..

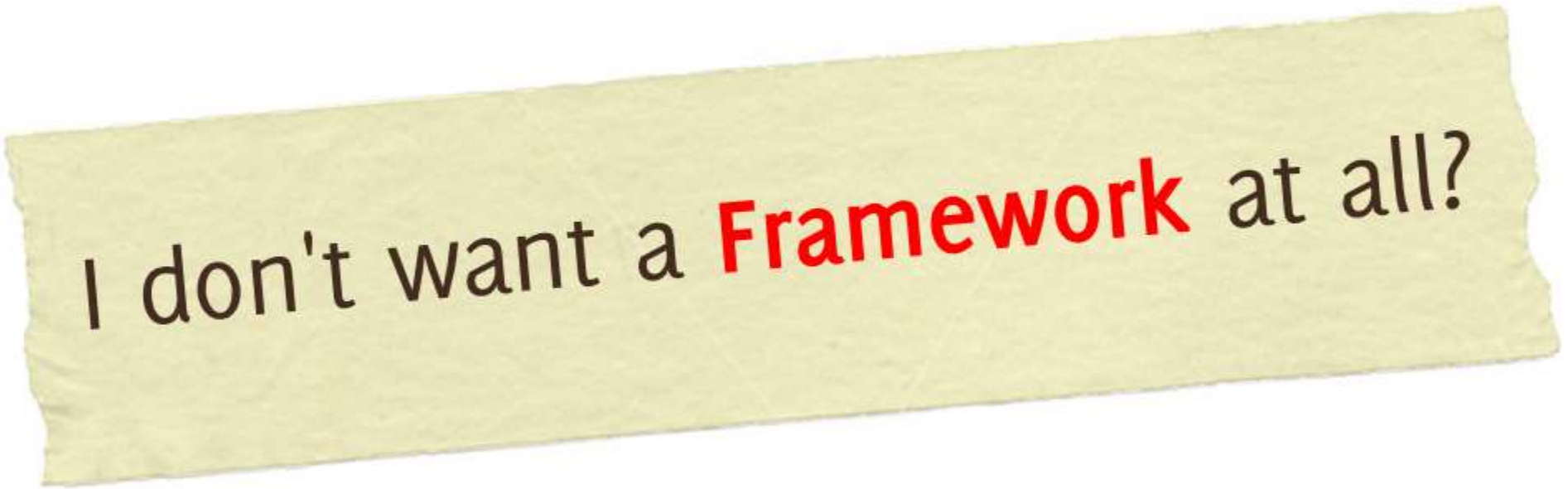
- Widget cross-platform frameworks (F

- Huge cross-platform frameworks
  (Qt/copperspice, wxWidgets, GTK+,

- Modern C++ focused on UIs (Nana C



Maybe…



An alternative approach

Integrate an **html UI** into my C++ desktop applications

Why html?

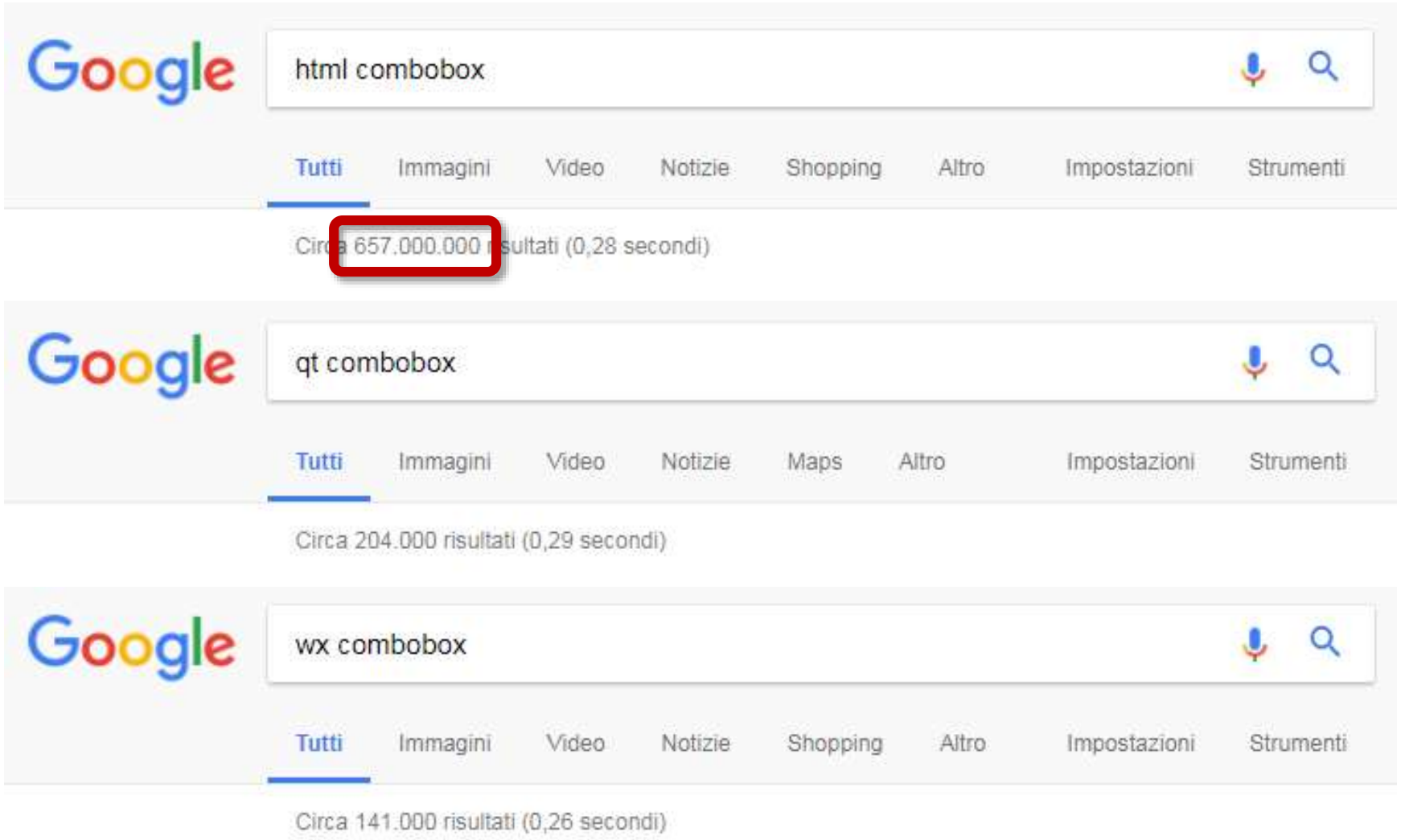
Better design

Graphic designers are used to html and css

Sharp division between layout and content (more or less…)

Lots of material and documentation

Lots of material and documentation



Not a silver bullet

It's not for all the applications

Just a solution I used many times in different circumstances

Previous attempts

Http (long polling) -> requires web server

Chrome extensions

Qt WebEngine

Electron / CEF (Chromium Embedded Framework)

Html5, finally!

Better graphics, multimedia, offline, storage, …

Above all:

Now we can load a local page interacting with a native application through **WebSockets**

Websocket

WebSocket is a computer communications protocol, providing full-duplex communication channels over a single TCP connection.

The WebSocket protocol enables interaction between a browser and a web server with lower overheads, facilitating real-time data transfer from and to the server. This is made possible by providing a standardized way for the server to send content to the browser without being solicited by the client, and allowing for messages to be passed back and forth while keeping the connection open. In this way, a two-way (bi-directional) ongoing conversation can take place between a browser and the server.

Websocket

WebSocket is a computer communications protocol, providing **full-duplex** communication channels over a **single TCP connection**.

The WebSocket protocol enables interaction between a browser and a web server with lower overheads, facilitating real-time data transfer from and to the server. This is made possible by providing a standardized way for the server to send content to the browser without being solicited by the client, and allowing for messages to be passed back and forth while keeping the connection open. In this way, a two-way (bi-directional) ongoing conversation can take place between a browser and the server.

Websocket

WebSocket is a computer communications protocol, providing full-duplex communication channels over a single TCP connection.

The WebSocket protocol enables interaction between a **browser and a web server with lower overheads**, facilitating real-time data transfer from and to the server. This is made possible by providing a standardized way for the server to send content to the browser without being solicited by the client, and allowing for messages to be passed back and forth while keeping the connection open. In this way, a two-way (bi-directional) ongoing conversation can take place between a browser and the server.

Websocket

WebSocket is a computer communications protocol, providing full-duplex communication channels over a single TCP connection.

The WebSocket protocol enables interaction between a browser and a web server with lower overheads, facilitating real-time data transfer from and to the server. This is made possible by providing a standardized way for the server to send content to the browser without being solicited by the client, and allowing for messages to be passed back and forth while keeping the connection open. In this way, a two-way (bi-directional) ongoing conversation can take place between a browser and the server.

websocket advantage #1

Browsers don't enforce the Same Origin Policy (and its CORS relaxation) for WebSockets as opposed to AJAX calls (*)

=> no webserver needed

(*) still, a websocket server can restrict access by checking "origin"

websocket advantage #2

Provides a standardized way for the server to send content to the browser without being solicited by the client.

The WebSocket protocol enables interaction between a browser and a web server with lower overheads, facilitating real-time data transfer from and to the server.

=> Solves the problem of pushing events to the client

#1: bring in websockets

Client (javascript): overly simple

```
var ws = new WebSocket( 'ws://localhost:9971' );

ws.onmessage = function(msg) { console.log(msg); }
ws.onopen = function () { console.log( 'WS connected' ); }
ws.onclose = function () { console.log( 'WS closed' ); }
ws.onerror = function () { console.log( 'WS error' ); }

…

if (ws.readyState == ws.OPEN) ws.send( 'hello world!' );
```

#1: bring in websockets

Server (C++): websocket library

boost::beast

- asynchronous (foster single thread)
- boost::asio paradigm (proactor and reactor)
- coming to Boost 1.66.0
- interface a little too much error-prone for my taste ☺

#2: choose a data serialization format

Json is the format with less friction for Javascript

```
var obj = {
   foo: 'foo string',
   bar: 42
};

var jsonString = JSON.stringify( obj );      // serialize

var jsonObj = JSON.parse( jsonString );   // deserialize

assert( jsonObj.foo === 'foo string' );
assert( jsonObj.bar === 42 );
```

#2: choose a data serialization format

On the C++ side: libraries to convert data <-> JSON

(e.g., nlohmann json)

```
json j2 = {
  {"pi", 3.141},
  {"happy", true},
  {"name", "Niels"},
  {"nothing", nullptr},
  {"answer", {
    {"everything", 42}
  }},
  {"list", {1, 0, 2}},
  {"object", {
    {"currency", "USD"},
    {"value", 42.99}
  }}
};
```

```
// create object from string literal
json j = "{ \"happy\": true, \"pi\": 3.141 }"_json;

// or even nicer with a raw string literal
auto j2 = R"(
  {
    "happy": true,
    "pi": 3.141
  }
)"_json;

// explicit conversion to string
std::string s = j.dump();    // {\"happy\":true,\"pi\":3.141}
```

Choices, as usual

We need to decide which part goes into the C++ component and which into the Javascript client

Not so bad…

We're forced to do something we usually don't do with classes inside the same component:

Carefully design the interface between the two components.

First try: Dumb Client

The protocol objects are the html graphical widgets

Events (js -> C++): "button X has been pressed"

Commands (C++ -> js): "set the text of label Y to Z"

Javascript client is a dumb proxy

Packaging…

We want the usual desktop application behaviour.

=> just add a script that launches the .exe and then the browser

```
REM file softphone.cmd

@echo off
cd content
start "" ui_websocket.exe
start "" index.htm
exit
```

… with a server shutdown when the connection is lost.

C++ side

```
void  OnIncomingCall()
{
   using json = nlohmann::json;
   json msg = {
      { "id", "statuslabel" }
      { "request", "setlabel" },
      { "value", "Incoming" }
   };
   webSocket.Send( msg.dump() );
}
```

Javascript side

```
webSocket.onmessage = function (event) {
   var jsonObj = JSON.parse(event.data);
   var item =
      document.getElementById(jsonObj.id);
   switch (jsonObj.request) {
      case "setlabel":
         item.innerHTML = jsonObj.value;
         break;
       case "setprogressbar":
         item.style.width = jsonObj.value;
         break;
       …
   }
};
```

Javascript side

```
function onCallClick() {
   var ev = {
      event: 'onclick',
      source: 'callbtn'
   };
   webSocket.send( JSON.stringify(ev) );
}

function onHangupClick() {
   var ev = {
      event: 'onclick',
      source: 'hangupbtn'
   };
   webSocket.send( JSON.stringify(ev) );
}
```

C++ side

```
webSocket.OnMessage( [&](const string& msg){
   using json = nlohmann::json;

   auto jsonObj = json::parse(msg);
   auto source = jsonObj.at("source");
   auto event = jsonObj.at("event");

  auto handler =
      handlers.find( make_pair(source, event) );
   if (handler != handlers.end()) handler->second();
} );
```

Improvement #1

```
void OnIncomingCall()
{
   webSocket.Send("document.getElementById('statuslabel').innerHTML='Incoming'");
}
```

```
webSocket.onmessage = function (event) {
   eval( event.data );
};
```

Improvement #2

```
$(function () {
  // all the buttons
  var buttons =
    document.getElementsByTagName('button');
  for (var i = 0; i < buttons.length; i++) {
      var button = buttons[i];
      button.onclick = function () {
          const ev = {
              event: 'onclick',
              id: this.id
          };
          webSocket.send( JSON.stringify(ev) );
      };
  }
  // same for inputs etc.
}
```

```
webSocket.OnMessage( [&](const string& msg){
   using json = nlohmann::json;

   auto jsonObj = json::parse(msg);
   auto source = jsonObj.at("source");
   auto event = jsonObj.at("event");

   auto handler =
      handlers.find( make_pair(source, event) );
   if (handler != handlers.end()) handler->second();

} );
```

Can you spot the asimmetry?

It seems like we can register javascript callbacks to C++ events

But not the other way around!

The asimmetry is due to languages differences:

- Javascript eval can exec an arbitrary string

Are we done, then?

… not really.

All the work is done by C++ (even input validation!)

It's far better if the protocol speaks in terms of domain (e.g., phone call, ringtone, speaker, microphone,…):

- input validation in javascript

- better separation between app logic and UI

- app logic independent from the UI representation

An higher level protocol…

Direction: UI javascript -> C++ Application

```
{
   "request": "connection",
   "data": {
      "server": "127.0.0.1",
      "user": "daniele",
      "password": "123456"
   }
}

{
   "request": "call",
   "data": {
      "number": "338123456"
   }
}
```

Direction: C++ Application -> UI javascript

```
{
   "event": "levelschanged",
   "data": {
      "miclevel": 32,
      "speakerlevel": 73
   }
}

{
   "event": "regstatuschanged",
    "data": {
      "status": "unregistered"
   }
}
```

Too much code to write?

```
// QT code for event handling

void Ui::AutoanswerActive(bool active)
{
   softPhone.Autoanswer(active));
}
connect(autoanswerCheckBox, SIGNAL(clicked(bool)), this, SLOT(AutoanswerActive(bool)));
```

```
// equivalent code

handlers["autoanswer"] = [&](json event){
    bool active = event.at("active");
    softPhone.Autoanswer(active);
});
```

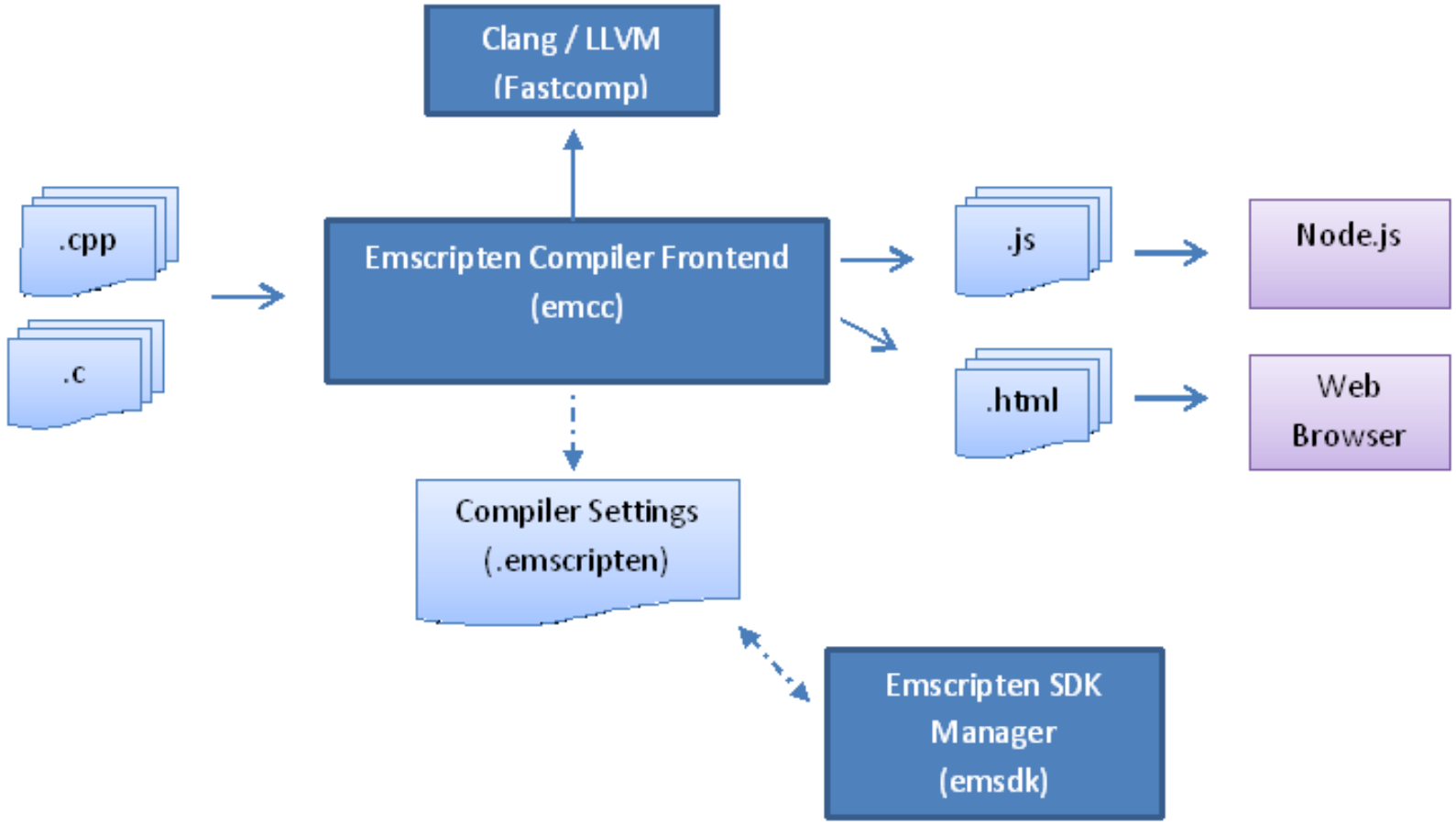
Too much code to write?

```
using json = nlohmann::json;

std::unordered_map<std::string, std::function<void(json)>> handlers;


webSocket.OnMessage( [&](const string& msg){

    auto jsonObj = json::parse(msg);
    auto event = jsonObj.at("event");

    auto handler = handlers.find( event) );
    if (handler != handlers.end()) handler->second(jsonObj);
} );
```

"...but javascript sucks!"

... you can even use C++ inside the browser!

Emscripten



emcc is a LLVM back end that produces *asm.js*

asm.js

Intermediate programming language intended to simulate Assembler.

It's a small subset of JavaScript optimized for performances.

Runs everywhere, but all the four major browsers optimize for asm.js

```
function Vb(d) {
  d = d | 0;
  var e = 0, f = 0, h = 0, j = 0, k = 0, l = 0,
      m = 0, n = 0, o = 0, p = 0, q = 0, r = 0, s = 0;
  e = i;
  i = i + 12 | 0;
  f = e | 0;
  h = d + 12 | 0;
  j = c[h >> 2] | 0;
  if ((j | 0) > 0) {
    c[h >> 2] = 0;
    k = 0
  } else {
    k = j
  }
  j = d + 24 | 0;
  if ((c[j >> 2] | 0) > 0) {
    c[j >> 2] = 0
  }
  ...
}
```

Emscripten APIs supported

C++ standard libraries which don’t need to interact with the system

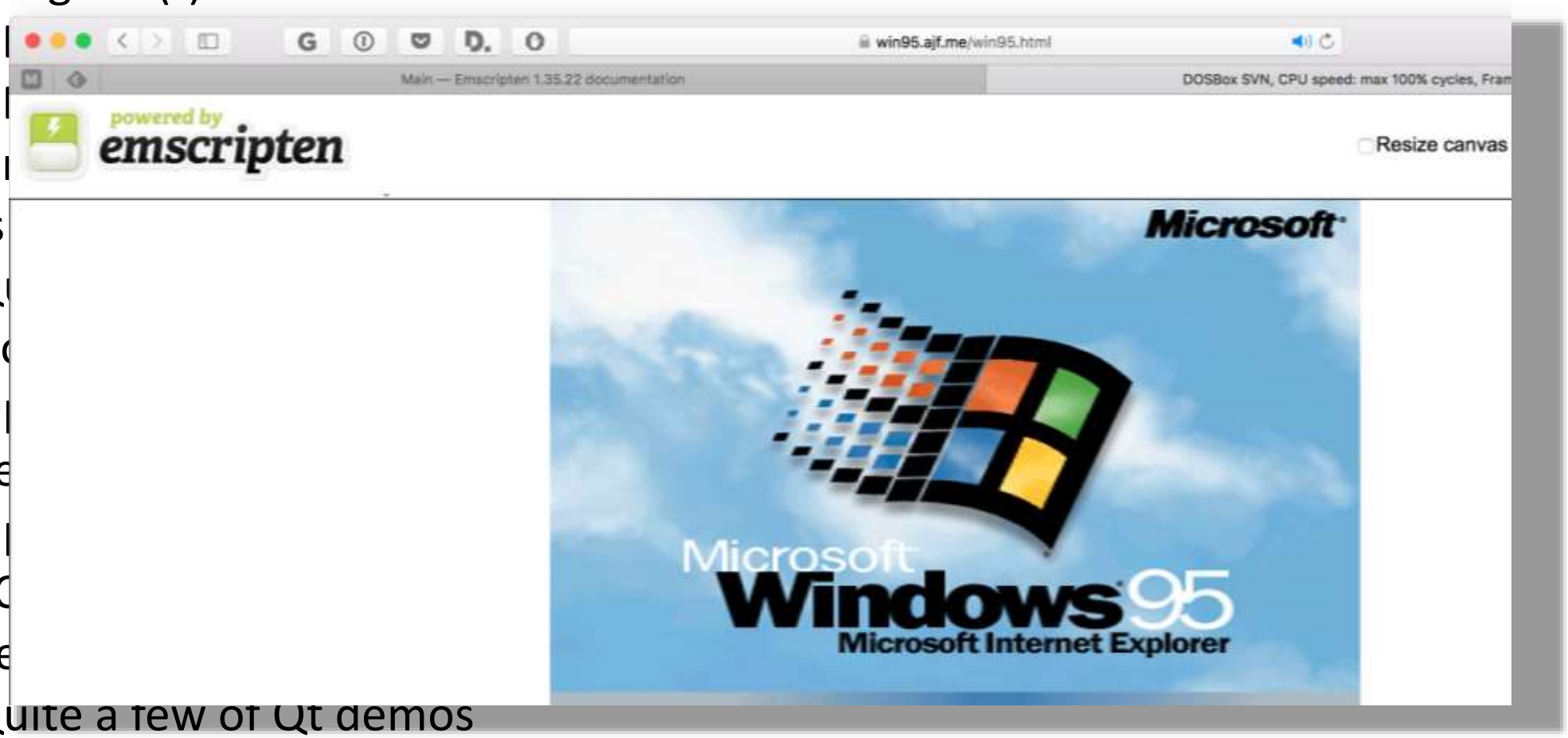
- Network support (libc-style, non-blocking only(!))
- File system access
- Graphics (OpenGL ES)
- Audio, keyboard, mouse, joystick (SDL)
- Integration with HTML5

Emscripten usage

- Game Engines(!)
  - UE3 (reported to be ported in 4 days)
  - UE4
  - Unity (C# to C++ with IL2CPP and C++ to asm.js with Emscripten)
- Games
  - Quake 3
  - Doom
  - OpenDune
- Libraries/Frameworks
  - OpenSSL
  - SQLite
  - Pepper (via pepper.js)
  - Quite a few of Qt demos

Emscripten usage

- Game Engines(!)
  - Ul
  - Ul
  - Ur
- Games
  - Qu
  - Do
  - Op
- Librarie
  - Op
  - SC
  - Pe
  - Quite a few of Qt demos



Does it make sense?

> We have different materials, like statically typed and dynamically typed languages, yet we don't have a sound theory of forces, so we end up with the usual evangelist trying to convert the world to its material of choice.
>
> It's kinda funny, because it's just as sensible as having a Brick Evangelist trying to convince you to use bricks instead of glass for your windows ("it's more robust!"), and a Glass Evangelist trying to convince you to build your entire house out of glass ("you'll get more light!").

Carlo Pescio

Let's recap

A "one size fits all" approach never works

It's just a new way you can take into account:

- Crossplatform (as much as the C++ you write)
- No constraint on your C++ code (no hugly macros, no custom compilers, …)
- Sharp separation between UI and app logic
- Multiple skins
- Automatic test of the whole application (plugging a test websocket client)
- You can use the whole toolset available for JS (people included)
- Idea useful also in other languages (see scripting languages)

But – please – split the application the right way!

References


Me: @DPallastrelli


Me: it.linkedin.com/in/pallad


Github: http://github.com/daniele77