C++ everywhere & in the Browser

sauter[devs] – 19. May 2015

Damien Buhl alias daminetreg

Agenda

- C++ everywhere ? Why ?
- Cross-platform source code, what does it mean ?
- Bringing the same code everywhere
- Notably In the browser
  - Simply running an hello world
  - Bringing a C++ library to the web
  - Calling JS Back from your C++
- How usable it is today
- How unusable it is today

C++ everywhere ? Why ?

- Best known cross-platform language
- Non-runtime dependent language, that means it can run under any runtime (JVM, .NET, Javascript VM, bare-metal, Linux, Windows, Mac process, Google Pnacl...)
- Strongly, statically typed, statically asserted, type-safe language
- Extremely backward compatible between decades
- Extensible language via meta-programs (DSELs, language extensions, compile-time calculations...)
- Open Standard by ISO, supported actively by all the big concerns

  C++98, C++11, C++14, C++17, C++ 21...

Cross-platform source code, what does it mean ? - I

- The same code, tested on many platform, recompiled to the platform specific runtime

- Code which uses STL, Boost and any other platform abstractions

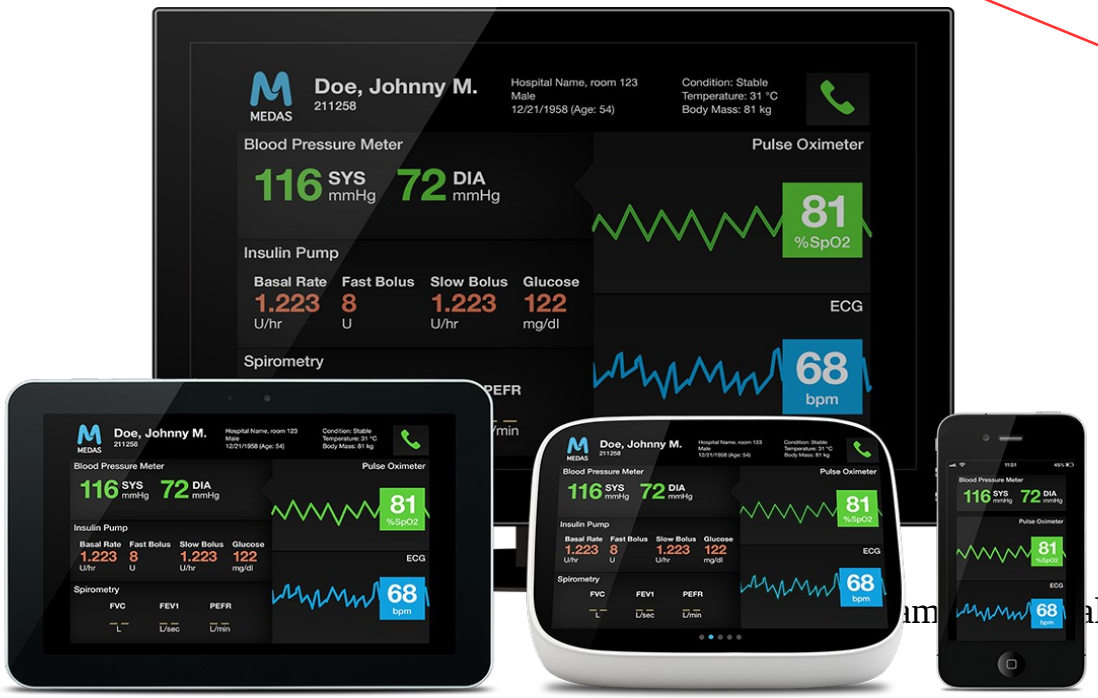
- Not a cross-platform binary
  - Advantages :
    - It really work on any platform and early when platform appear.
    - Doesn't has the performance penalty of an interpreter / JITter
    - Can still run within another interpreter / JITter when compiled for it.
  - Disadvantages :
    - Needs to recompile for the different platforms and ship different binaries.

Cross-platform source code, what does it mean ? - II

- Cmake
  - Toolchain files
- STL + Boost
- Building up abstractions for platform specific access not covered by the STL or Boost
- Fine tuning for each platforms
  - Only if high performance needed
  - For some specific behaviour on the different platforms
- A portable GUI toolkit
  - Qt, wxWidgets ?
- “A program that has not been tested does not work.” - *Stroustrup, Bjarne. The C++ Programming Language. pp. 712.*

Bringing the same code everywhere

- In Any OS
  - Windows, Linux, Mac OS X
  - Iphone, Android, Windows Phone 8
  - On any browser



Same Hello World everywhere

Demo

LLVM + Emscripten - I

- LLVM backend compiling LLVM bitcode into Javascript
  - Namely into standardized asm.js
  - "*The asm.js programming model is built around integer and floating-point arithmetic and a virtual heap represented as a typed array.*" - asm.js specification, August 2014

- Allows using clang C++ compiler and any language which compiles into LLVM bitcode in a Javascript runtime.

LLVM + Emscripten - II

- It's also an API and lower-level libraries ported to the browser :
  - POSIX file + network implemented via Javascript bindings
  - SDL & OpenGL thanks to Canvas & WebGL
  - Libc & STL C++ library
  - X11 windowing library for linux
  - Audio: OpenAL ported via javascript Media API

Bringing a C++ library to the web - I

- An example : AlarmWidget
  - Can render 1 alarm
  - Handle click events to move to next alarm
  - Load alarm from an array

- Technology :
  - It renders it with the SDL library
  - Use plain standard C++11
  - Uses Boost

Bringing a C++ library to the web - II

- Standard Type conversion between emscripten C++ HEAP & Javascript
  - e.g. std::string is converted to a javascript string filled with one 32bit integer for the length + the char buffer

- Normal stack & HEAP in emscripten are handled with Javascript typed arrays, that is the types like integer, float... are exactly the same between C++ & JS and just need accessing an index of the array behind the scenes.

Bringing a C++ library to the web - III

- Embind
  - Similiar API as Boost.Python
  - Allows to define which functions/classes are to be accessible as a normal Javascript function
  - Features mapping of types, classes, enums, shared_ptr, containers...
    - std::string maps to String
    - Int,float,double... to Number
    - bool to true/false
    - void to js undefined

Calling JS back from C++

- Possibility to subclass C++ mother class in Javascript (via virtual inheritance, C++ will call your javascript method)
- Possibility to use any of the Browser js API via emscripten::val.
- 200ns overhead measured for JS call from C++

How usable it is today

- Lifecycle spoken : in growth near to maturity
- It is faster than writing javascript yourself as it uses the standard : asm.js
- It's fair to say that it is 50% the same C++ code compiled native
- Debugging with source maps is working well
- Binding between JS Handwritten code and C++ is well handled and can be fine tuned

How unusable it is today

- Tooling is not perfect as with native cross-platform development

- There is possibly a need for a more modular GUI framework
  - Qt is too monolithic in this regard
  - SDL / openGL are only about rendering
  - What about building up Boost.UI together ?

References

- http://emscripten.org/
- http://kripken.github.io/emscripten-site/docs/porting/connecting_cpp_and_javascript/embind.html#embind
- http://www.slideshare.net/chadaustin/connecting-c-and-javascript-on-the-web-with-embind