Current Trends in Web Engineering

Prof. Dr.-Ing. Martin Gaedke
Dr.-Ing. Sheeba Samuel

Technische Universität Chemnitz

Fakultät für Informatik

Verteilte und selbstorganisierende Rechnersysteme

CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz

Part III

Advanced Aspects

Lecture Outline

Current Trends in Web Engineering (Advanced Web Engineering Aspects)

1 Development	2 Social Web	3 Advanced Aspects
Agile Approaches	P[illegible]tforms	Current trends 1
Design Thinking	Identity & more	Current trends 2
Lean Development	Frameworks & more	Current trends 3

We are here

CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

Chapter://1

Web Assembly

WebAssembly (WASM)

- W3C Standard defining a binary instruction format (bytecode) for execution within web browsers
- https://www.w3.org/TR/wasm-core-1/
- Enables
  - Higher performance compared to JS
  - Usage of non-JS languages in browser

CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

WebAssembly Goals

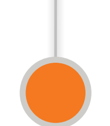
- Efficiency and Portability
  - Near-native execution speeds across different platforms
- Readability and Debuggability
  - specifies additional human-readable text format
- Security
  - sandboxed, SOP and permission policies enforced
- Open Web Platform Integration
  - Interaction to/from JS through JS Interface
  - Using Web APIs (DOM, WebSockets, IndexedDB, etc.)
  - W3C Standard

CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

WebAssembly Predecessors

- NaCl: Google Native Client
  - Sandbox for running compiled C/C++ code in the browser
  - Interaction with JS environment
  - Compilation to bytecode (PNaCl)
  - Browser support limited to Chrome
  - deprecated in 2020
  - https://developer.chrome.com/docs/native-client/
- asm.js
  - Strict subset of JavaScript as compilation target for statically-typed languages, targeting high-performance applications (e.g., Game Engines like Unity, Unreal were ported)
  - Natural interaction with JS environment
  - Compilers like emscripten/cheerp can compile C/C++ to asm.js
  - Initially supported by Mozilla
  - Obsoleted by advent of WASM, sometimes as „fallback“
  - http://asmjs.org/spec/latest/

CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

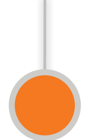


WebAssembly vs JavaScript?

Web Browser

VM (Engine)	Web APIs	Browser/Device Functionalities
JS	DOM	Screen/Input
⇕	Web Sockets	Network
WASM	LocalStorage	Storage
	MediaStreams	Camera
	…	…

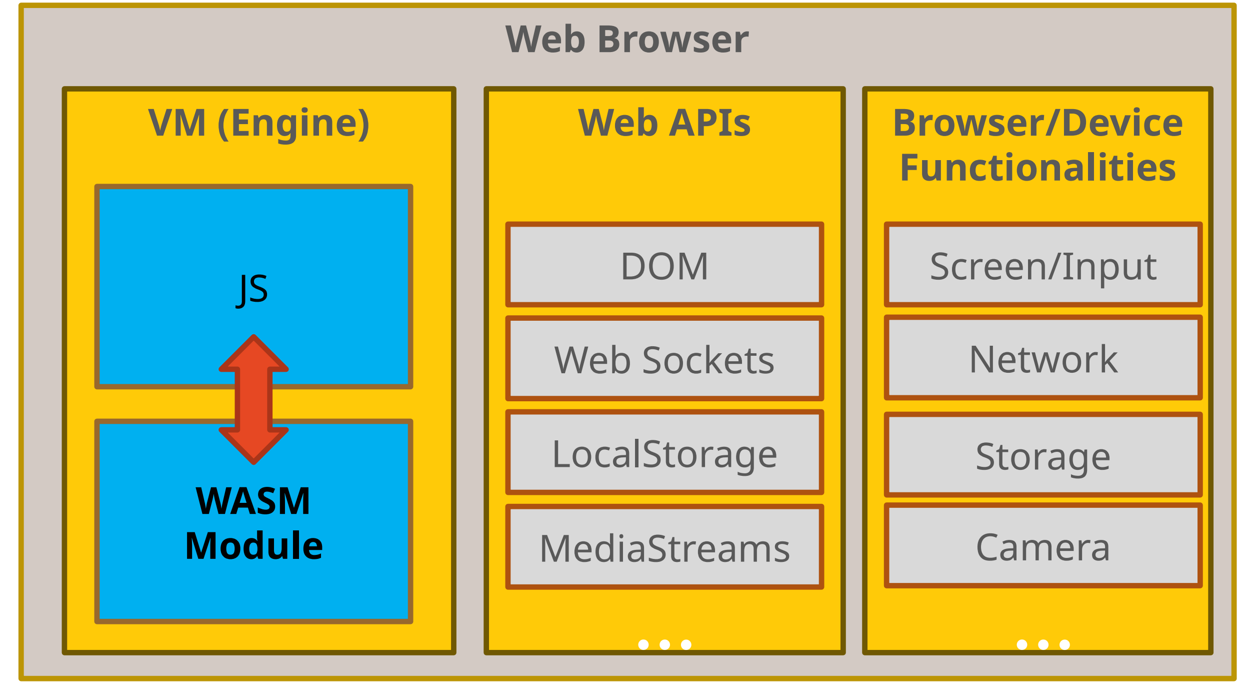
CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE



WASM Concepts: Module

WASM Module

- a binary compiled into machine code executable in the stack-based WASM VM



CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

WASM Concepts: Memory

Memory

- Resizable ArrayBuffer for reading/writing bytes
- Accessible from JS and WASM code

```
01  var memory = new WebAssembly.Memory({initial:10, maximum:100});
02
03  WebAssembly.instantiateStreaming(fetch('memory.wasm'), { js: { mem: memory } })
04    .then(obj => {
05        var i32 = new Uint32Array(memory.buffer);
06        for (var i = 0; i < 10; i++) {
07          i32[i] = i;
08        }
09        var sum = obj.instance.exports.accumulate(0, 10);
10        console.log(sum);
11   });
```


CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

WASM Concepts: Table

Table

- Resizable array storing function references
- Allowed element type is `anyfunc`
- `call_indirect` for function calls

```
(type $FUNCSIG$i (func (result i32)))
(table 1 anyfunc)
(elem (i32.const 0) $test)
        …
(func $test (type $FUNCSIG$i) (result i32)
  (i32.const 42)
)

(func $main (result i32)
  (call_indirect $FUNCSIG$i
    (i32.const 0)
  )
)
```

Function Signature Declaration

Table Declaration,
adding function $test at index 0

Indirect function call using
index 0

CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

WASM Concepts: Instance

Instance

- Module + Runtime State
- Includes a Memory, a Table and Imported Values

```
const imports = {
  imports: {
    imported_func: function(arg) {
      console.log(arg);
    }
  }
};

WebAssembly.instantiateStreaming(fetch('test.wasm'), imports)
.then(obj => obj.instance.exports.exported_func());
```


CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

WASM Text Format

- Textual representation of binary wasm
- Can be converted to and from binary wasm
- Useful for debugging, learning, writing by hand, viewing source of modules
- `.wat` files encoded in UTF-8
- Syntax based on S-expressions

CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

WASM Text Format Example

```
1 (module
2    (func $i (import "imports" "imported_func") (param i32))
3    (func (export "exported_func")
4      i32.const 42
5      call $i
6    )
7  )
```


CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

WASM Format Conversion

- WABT (WASM Binary Toolkit)
- convert text format to wasm binary: wat2wasm
- convert wasm binary to text format: wasm2wat
- Online Demos available
  - https://webassembly.github.io/wabt/demo/

CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

WASM Compilation

General Idea

Examples

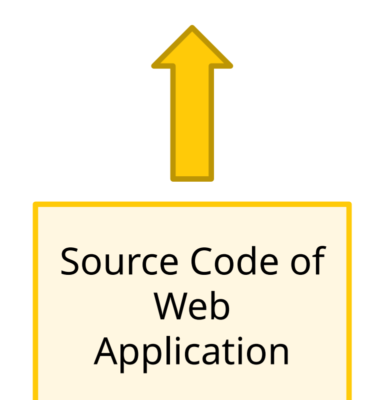
- C/C++: emscripten, cheerp via LLVM
- Go: native support since Version 1.1
- Rust: wasm-pack
- Kotlin: Kotlin/Native via LLVM

CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

WASM Compilation

What about *interpreted languages* like

python, ruby, PHP, *JS*?

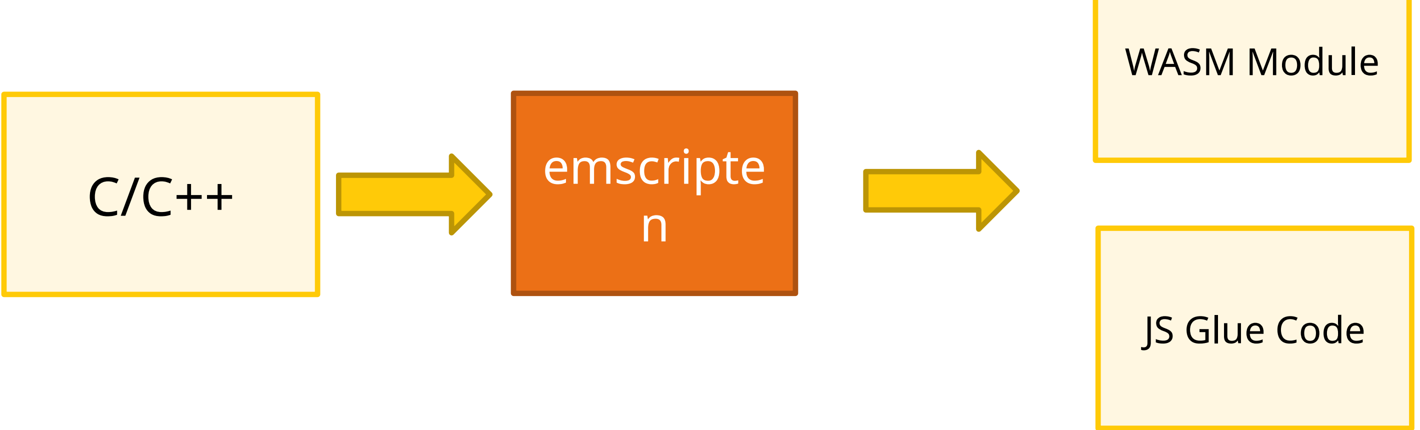


Examples:

- python: pyodide
- Ruby: prism via mruby/emscripten

CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

Emscripten



JS Glue Code

- Loader for WASM Module
- Generated JS Code for support communication between native C/C++ functions and JS
- E.g., memory management, serialization of strings for message exchange

CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

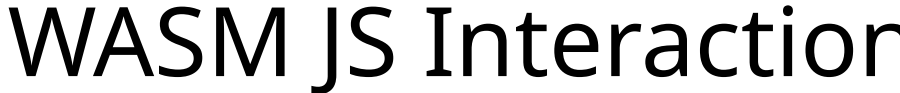
WASM JS Interaction

Calling functions within WASM Modules from JS

- Functions need to be declared as exported functions, e.g., using EMSCRIPTEN_KEEPALIVE in C/C++ with emscripten
- Emscripten will optimize out any non-declared functions apart from main

```
1 #include <stdio.h>
2 #include <emscripten/emscripten.h>
3
4 . . .
5
6 EMSCRIPTEN_KEEPALIVE int FunctionWithinWasm(int a, int b) {
7       return a + b;
8 }
9
```


CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE



Calling functions within WASM Modules from JS

- Compile with options `NO_EXIT_RUNTIME` to avoid runtime shutdown when `main()` exits and `EXPORTED_RUNTIME_METHODS` to export `ccall()`

```
emcc -o sample.html sample.c -O3 -s WASM=1 \
–s NO_EXIT_RUNTIME=1 \
-s "EXPORTED_RUNTIME_METHODS=['ccall']"
```


CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

WASM JS Interaction

- From JS, use ccall() to call the exported function

```
<script src="sample.js"></script>
<script>
    function callToWasm() {
            const result = Module.ccall('FunctionWithinWasm',
             'number', // return type
             ['number', 'number'], //argument types
             [1, 2]); //arguments

            console.log(result);
    }
</script>
```


CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

WASM JS Interaction

- Use cwrap() to generate a JS function wrapping the exported function

```
1   <script src="sample.js"></script>
2   <script>
3       function callToWasm() {
4               function_wrapper =
Module.cwrap('FunctionWithinWasm',
5                'number', // return type
6                ['number', 'number'], //argument types
7                ); //no arguments this time
8
9               console.log(function_wrapper(1, 2));
10      }
11  </script>
```


CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

WASM JS Interaction

Calling functions within JS from WASM modules

```
1  <script>
2      function FunctionWithinJs(a, b) {
3              return a + b;
4      }
5  </script>
```


CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

WASM JS Interaction

- Use EM_JS() to call JS functions from within WASM

```
1  #include <emscripten.h>
2
3   EM_JS(int, // return type
4     JsWrapper, // function name
5     (int a, int b), // arguments
6     {
7     // arbitrary JS code
8     // can call external JS functions
9     return FunctionWithinJs(a, b);
10   });
11
12  int main() {
13    int result = JsWrapper(1, 2);
14    return 0;
15  }
```


CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

WASM JS Interaction: Types

- Only numerical values can be exchanged between JS and WASM (numbers, pointers)
- WASM and JS use different type systems
- Number formats (in-memory representations) are different
  - WASM: int32, int64, float32, float64
  - JS: Number
- JS API Type coercion methods for numbers
  - ToJsValue()
  - ToWebAssemblyValue()

CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

WASM JS Interaction: Types

- Non-numerical types need to be represented as numbers
- Example: passing a string between JS and WASM
  - String → array of numbers → String
  - JS Glue Code offers support for these conversions
    - intArrayFromString() converts JS String to C-line array of numbers 0-terminated
    - intArrayToString() converts 0-terminated C-line array of numbers to JS String

CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

WASM JS Interaction: Types

- Non-numerical types require memory management
  1. Creating a pointer on the WASM Heap
  2. Copying the value into memory
  3. Passing pointer to WASM
  4. Freeing the memory
- ccall() and cwrap() handle memory management for string and array (only of 8-bit values) types for convenience
  - but memory is immediately freed after function call
  - stored pointers inside WASM Module may point to invalid data afterwards
- Persistent objects require manual memory management using _malloc() and _free()

CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

WASM JS Interaction: Types

- Example for manual memory management, passing an array of integers (i.e., 32-bit values) to WASM

```
1  const array = [1, 2, 3, 4];
2  const length = array.length;
3  const bytesPerElement = Module.HEAP32.BYTES_PER_ELEMENT;
4
5  const arrayPointer = Module._malloc((length * bytesPerElement));
6
7  Module.HEAP32.set(array, (arrayPointer / bytesPerElement));
8
9  Module.ccall(‚WASMFunctionReceivingArray',
10         null,
11         ['number', 'number'],
12         [arrayPointer, length]);
13
14 Module._free(arrayPointer);
```


CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

WASM Dependencies

- Any dependency code inside WASM needs to be either
  - Available as source code and compilable to WASM
  - Part of the standard library/environment of the particular language (e.g., from emscripten for C/C++, from Go environment for Go etc.)
  - (Available as a WASM module – ongoing work on the Module Linking Proposal)
- If unavailable
  - Own re-implementation may be required
  - Some functionalities (e.g., DOM access, Browser APIs etc.) may be available in JS, so a possible workaround is to call to JS from within WASM module

CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE

Discussion

- Which current use cases do you see for using WASM?
- WASM does not explicitly aim to replace JS. Assume it would. What would that enable and what would be the benefits?

CC-BY-NC: Prof. Dr. Martin Gaedke, Dr. Sebastian Heil · Professur VSR · Fakultät für Informatik · TU Chemnitz . Lecture CTWE