Benjamin Canou, Emmanuel Chailloux and Jérôme Vouillon

How to Run your Favorite Language on Web Browsers

The Revenge of Virtual Machines

Paris, le 4 octobre 2011

WWW 2012, Lyon, France

Introduction

What ?

- You have a favorite language
- You have just designed or extended one
- You want to run it on a Web browser

Why ?

- To program a new Web app
- To program your client with the same language than your server
- To run an online demo of an existing app

How ?

- Use applets
- Write an interpreter in JavaScript
- Write a compiler to JavaScript

Or as we present in this talk:

- Reuse the language bytecode compiler
- Write a bytecode interpreter in JavaScript
- Write a bytecode to JavaScript expander

An experiment report:

- Project Ocsigen: use OCaml to code entire Web apps
- OBrowser: an OCaml bytecode interpreter
- js_of_ocaml: an OCaml bytecode expander

Retrospectively, a good approach:

- Reasonable time to obtain a first platform
- Good performance achievable
- Fidelity to language/concurrency models

Core techniques

Main method:

1. Make the bytecode file network compliant (ex. JSON array)
2. Choose/implement the representation of values
3. Write a minimal runtime and standard library
4. Write the main interpretation loop
5. Run tests and extend the library as needed

Possible improvements:

- Use core, well supported/optimized JavaScript structures
- Use simple, array based memory representation
- Preliminary bytecode cleanup pass

Pros:

- Fairly simple architecture
- Debug/adjustments using step-by-step execution
- The original VM can be used as a reference
- Semantics preservation
- Acceptable performance

Cons:

- Impossible to obtain great performance

Experiment: OBrowser

- Bytecode for the OCaml virtual machine
- A few weeks to develop and debug
- Performance < 10x JavaScript equivalents
- Runs existing OCaml programs, compiled with unmodified ocamlc
- Actually usable to start writing Web apps in OCaml

Demo: a *Boulder Dash* clone

- Uses the DOM and HTML elements for the interface
- Event handlers in OCaml
- Loads levels via HTTP requests
- In pretty standard OCaml style

Basic method:

1. Reconstruct the control flow graph
2. Expand every basic block to a JavaScript function
3. Expand every bytecode to javascript instructions

Necessary improvements (for code size:

- Intra-procedural expression reconstruction
- Dead code elimination

Possible improvements:

- Finer (than function only) basic block mapping
- Run-time inlining
- Any compiler optimization

Pros:

- Potential great performance
- Easier to write than a from-source compiler
- Lower maintenance cost than a from-source compiler

Cons:

- More difficult to write than an interpreter
- Takes more time to see your first program running
- Easier to introduce bugs/more difficult to debug

Experiment: js_of_ocaml

- Compiles OCaml bytecode to JavaScript
- Runs existing OCaml programs, compiled with unmodified ocamlc
- Excellent performance
- A few concessions to semantics preservation

Demos:

- Real time 3D software rendering
- OCaml compiler and interactive prompt
- An SMT solver in the browser !

	Compiler	VM	Expanser
Simplicity	+	++	+
Semantics preservation	++	+++	++
Maintenance	+	+++	+++
Performance	+++	+	++
Concurrency	++	+++	+

1. Write a bytecode interpreter
2. Start writing a bytecode expander if performance is required
3. When the interpreter is ready, you can start developing your Web app
4. Use the expander in production
5. The interpreter can be used for debugging

Advanced topics

- Mutable strings

Conclusion