SPL and WebSPL

Yet another programming paradigm

Clifford Wolf - www.clifford.at
Introduction
Overview

- SPL is an embeddable scripting language with C-like syntax.
- It has support for arrays, hashes, objects, perl regular expressions, etc. pp.
- The entire state of the virtual machine can be dumped at any time and execution of the program resumed later.
- In SPL there is a clear separation of compiler, assembler, optimizer and virtual machine.
- It's possible to run pre-compiled binaries, program directly in the VM assembly, use multi threading, step-debug programs, etc. pp.
- SPL is a very small project, so it is a good example for implementing high-level language compilers for stack machines.
Aim

- Creating an interesting alternative to S-Lang for embeddable scripting languages.

- Creating an interesting alternative to Java and .NET for web applications.

- Creating a good example for simple and well designed virtual machine and a compiler for it.
Components

- SPL is a small library containing the SPL components:

- Virtual Machine: can execute SPL bytecode

- Assembler: can convert SPL assembler to bytecode

- Compiler: can convert SPL to SPL assembler

- Optimizer: plugs into the assembler and optimizes the bytecode

- Dumper: can dump and restore the VM state
Virtual Machine
Overview

- An SPL VM is basically the sum of: Codepages, Tasks (threads), Nodes (variables) and builtin functions

- Tasks have an instruction pointer and a (node) stack

- Nodes represent all kinds of variables and form a tree-like graph

- The builtin functions have a flat namespace and are global for the VM

- Almost everything else is pretty strict bound to some kind of context
Nodes

- Nodes are "the SPL variables".

- A node can hold various kinds of data:
  - Scalar
    - Integer, Float, String
  - Assoziative Arrays
  - Code Pointer
    - Functions, Return-Addresses, ...
  - Hosted Namespaces
  - Classes and Objects

- Each node has a context pointer and type
Architecture

- The SPL virtual machine is a simple stack machine
- It is using a hybrid reference counting mark recursive garbage collector
- The instruction set listing is in spl.h
- Implementation details: exec.c and state.c
The SPL Language
Overview

- SPL is syntactically a C-like language
- It has some concepts from JavaScript, Perl, OCaml and Nasal
- And some new concepts (at least I think so ;-)
- SPL is entirely typeless
- See code examples ...
Basics (1/2)

- Variables must be declared with the `var` keyword.

- `if`, `for`, `while`, `return` and so on work as in C.

- `foreach i (array) array.[i] = 42;` iterates over the keys of an associative array.

- `undef` is a constant expression for an undefined scalar value.

- `defined` is a check if the scalar value of a variable is defined.

- `declared` checks if a variable-name (key in associative array) exists.

- `delete` deletes an entry in an associative array.
Basics (2/2)

- The `asm` statement can be used to insert assembler code.

- `debug` can be used to write to the console.

```plaintext
function add3(a, b, c) {
  return a+b+c;
}
debug add3(10, 15, 20);
defines and calls a function.
```

- `foo = bar;` creates a copy for the node value and a reference for its childs (or at least it looks like that.. ;-)

- `foo := bar;` Create a real (recursive) copy of the node.
Operators

- !, | |, &&, not, and, or
  Logical operators

- ==, !=, <=, <, >, >=
  Integer comparisons

- .==, .=!, .<=, .<, .>, .>=
  Floating point comparisons

- ~==, ~!=, ~<=, ~<, ~>, ~>=
  String comparisons

- +, -, *, /, %, **
  Integer operators

- .+, .-, .*,. /, .%, .**
  Floating point operators

- ~
  String concatenation
In SPL, there is no difference between objects and classes. So it is called "objects" and "instances of objects".

```plaintext
object Foo {
    method init() {
        debug "Now in init() from A.";
        return this;
    }
}

object Bar Foo {
    method init() {
        debug "Now in init() from B.";
        return *A.foo();
    }
}

var foobar = new Bar();
```
Functions

- Functions (function pointers) are just variables.
- They can be copied, etc as any other variable.

While a function is executed, the parent context is the context in which the function has been defined, not the context from which the function has been called.

- `foobar(a, b, c);`
  calling a regular function or built-in function.

- `*foobar(a, b, c);`
  calling a function with current context as parent context.

- `$foobar(a, b, c);`
  calling a built-in function.
Here-documents

- `<< FOOBAR`
  - string until FOOBAR with $-Substitution

- `<<< this is test number $i`
  - string until EOL with $-Substitution

- `>> FOOBAR`
  - `>>> this is another test`
  - as above but without $-Substitution

- `<< FOOBAR:`
  - `>> FOOBAR:
    - as above but with indenting character`

- `<foobar>`
  - `</foobar>`
  - an inline template
Templates

- `<spl:if cond="defined userid">`  
  only include content if condition is true

- `<spl:foreach var="i" list="list">`  
  iterate over loop

- `<spl:code>`  
  execute embedded function and include return value

- `<spl:var name="query">`  
  set variable to content
Includes

- #file-as-const example11.txt
  insert file as string constant

- #file-as-code example11.code
  insert file as program code

- #file-as-template example11.tpl
  insert file as template with $-Substitution
  and interpretation of <spl:...> tags.

- #embedded-file demo.txt EOF .. EOF
  A "file" embedded like a here-document
  It can be accessed as *demo.txt
$ Substitutions

- **$variable**
  Insert value of variable

- **${var1 + var2}**
  Insert result of expression

- **$( if (x) return y; return "bla"; )**
  Insert return value of embedded function

- **$[ this is a comment ]**
  Comments in strings, templates or here documents
Advanced expressions

- **xml::variable**
  The same as  `encode_xml(variable)`

- `{ if (x) return y; return "bla"; }`
  An embedded function
The C-API
Overview

- Include SPL as vendor-branch in your apps.

- Two files: `spl.a` and `spl.h`

- Link apps with `spl.a` and `-rdynamic` (for module loading)

- SPL is very modular.
  - It is easy to take parts out or substitute them.

- All types and functions are prefixed with `spl_`
  - All preprocessor defines are prefixed with `SPL_`

- See: `spl.h` and `splrun.c`.
Simple example (1/2)

/* create VM and task structs */
struct spl_vm *vm = spl_vm_create();
struct spl_task *task = spl_task_create(vm, 0);

/* create assembler */
struct spl_asm *as = spl_asm_create();

/* compile and optimize */
char *spl_source = spl_malloc_file("example.spl", 0);
if ( spl_compiler(as, spl_source, "example.spl", spl_malloc_file) ) return 1;
free(spl_source);
spl_asm_add(as, SPL_OP_HALT, 0);
spl_optimizer(as);

/* dump bytecode to task, free assembler */
spl_task_setcode(task, spl_asm_dump(as));
spl_asm_destroy(as);
Simple example (2/2)

```c
/* register builtins to VM */
spl_builtin_register_all(vm);

/* runloop */
while ( task->code )
{
    /* handle scheduling */
    task = spl_schedule(task);
    if ( !task ) break;

    /* execute an instruction */
    int rlret = spl_exec(task);

    /* handle runtime error */
    if ( rlret < 0 ) return 1;
}

spl_vm_destroy(vm);
```
Extending SPL with additional builtin functions is easy. Here is an example from the XML module:

```c
struct spl_node *handler_encode_xml(
    struct spl_task *task, void *data)
{
    char *source = spl_clib_get_string(task);
    return SPL_NEW_STRING(xml_encode(source));
}

void spl_mod_xml_init(struct spl_vm *vm,
    struct spl_module *mod, int restore)
{
    spl_clib_reg(vm, "encode_xml",
        handler_encode_xml, 0);
}
Advanced examples

- Compiling and running SPL programs: splrun.c

- Implementing loadable modules: modules/mod_termio.c

- Implementing hosted namespaces: modules/mod_prime.c

- Embedding SPL bytecode in C programs: modules/mod_wsf.*
WebSPL, WSF, Tasks
WebSPL

- WebSPL is a framework for web application development.

- It creates a state over the stateless HTTP protocol using the dump/restore features of SPL.

- I.e. it is possible to print out an updated HTML page and then call a function which “waits” for the user to do anything and returns then.
WSF (WebSPL Forms)

- The DOM tree of a webpage is mapped to a tree of WSF objects.

- Each WSF object must implement the method `get_html()` which creates the DOM tree for the object and all its children as XHTML code.

- Each WSF object is running in its own task context.

- The WSF main loop is updating the webbrowser using a `JavaScript-in-IFrame-Hack` or by reloading the entire site.

- See `wsfdemo/wsfdemo.webspl` for a WebSPL example with WebSPL Forms.
WSF Dialogs

- A generic component for "clicking together" WSF components.

- A WSF Dialogs can be stored and loaded as XML files.

- The component already is the editor.

- A member function can be used to switch the modes.

```javascript
load "wsf";
load "wsf_dialog";

var page = new WsfDocument();
page.root = new WsfDialog( undef );
page.root.set_edit_mode(1);
page.main();
```
WSF Edit

- A generic component for database fronted-like components.
- Generic load and store functions.
- A list of fields actually present in the html output.
- A child class for simple SQL frontend is also available.

```plaintext
object Pref WsfEditSql
{
    var sql_db = db;
    var sql_table = "users";
    var sql_key = "rowid";

    method get_html() {
        edit_init();
        return #file-as-template edit_pref.tpl;
    }
}
```
Tasks API

- A generic environment for multithreading and co-routines in SPL.

- The host app must support it by calling the `spl_schedule()` function.

- It provides functions for creating, destroying, stopping and waking up tasks.

- `spl_schedule()` does round-robin scheduling between the tasks.

- Frameworks for co-routines, etc. can easily be implemented in SPL.
Simple XML API

- A generic API for reading and writing XML Files.

  xmltree = xml2tree(xmlfile, èrror);
  read XML file (DOM parser)

  xmlfile = tree2xml(xmltree);
  create XML from SPL DOM tree

  encoded = encode_xml(plaintext);
  xml-encode plaintext

  encoded = xml::plaintext;
  as above, but using the :: syntax

- TODO: XPATH API for searching in xmltree objects
URLs and References
SPL-Based Projects

- BotHack
  *still in design phase*

- QCake (aka. "KCake")

- WebSPL
  [http://www.clifford.at/spl/](http://www.clifford.at/spl/)

- More to come?
The SPL Project:
http://www.clifford.at/spl/

Clifford Wolf:
http://www.clifford.at/

LINBIT Information Technologies
http://www.linbit.com/