Contents

1 Intro

Hello, my name is Dmitry Matveev, I am a GSoC 2010 student. I am working on the project named "Progress Towards a Cross-Dialect Smalltalk FFI" (`http://gsoc2010.esug.org/projects/ffi`).

I've read the CObject/Alien documentation and I've outlined the major FFI components. FFI should have tools for:

- representing the scalar C data types;
- representing the C data structures;
- C callouts.

2 Representing the scalar C data types

The FFI system should have a set of the classes representing the basic C scalar data types - integers, characters, etc. Each class has to know:

1. A size in bytes required to store the C variable of this type;
2. A way how to serialize the appropriate Smalltalk data type into the actual bytes acceptable by a C-side code;
3. A way how to build the appropriate Smalltalk variable from the actual bytes.

There should not be any problems, everything is obvious and it's easy to implement. GNU Smalltalk already has the set of such classes.

3 Representing the C data structures

The FFI system should have a routine for representing the complex C data structures in Smalltalk using the scalar data classes from the p.2. As far as I understand, in the Alien user has to create a class representing a structure, and

a set of acessors that will read/write the data from the raw byte array using the different offsets for the appropriate structure fields (please correct me if I'm wrong). In GNU Smalltalk there is a more convenient way of representing the C structures:

```
CStruct subclass: AudioInfo [
    <declaration: #((#play #{AudioPrinfo} )
                    (#record #{AudioPrinfo} )
                    (#monitorGain #uLong)
                    (#yyy (#array #uLong 4)))>

    <category: 'C interface-Audio'>
]
```

This example is taken from the GNU Smalltalk manual. The `ClassDescription` subclass is used here to specify the structure representation via the `<declaration:>`; the appropriate field accessors will be generated automatically using it. I like this way and I am sure that it's possible to implement the same approach in Squeak.

4 C callout mechanism

GNU Smalltalk (and the FFI for Squeak) both use the primitive-like syntax, i.e:

```
ExternalInterface >> system: aString
  <cCall: 'system' returning: #int args: #(#string)>
```

The code is self explanatory. Broadly speaking, the following actions are performed here in my understanding:

1. The virtual machine lookups the address of the C function by its name (via the `dlsym()` in Unix or the `GetProcAddress()` in Windows) in all the loaded dynamic libraries (`dlopen()`/`LoadLibrary()`);
2. The virtual machine parses the argument string, extracts the method parameters and converts it into the plain C format according to the specified types;
3. The virtual machine pushes the arguments into the stack (platform-dependent operations here, libffi rocks) and calls the function by it's address;
4. The virtual machine converts the return value to the Smalltalk type and returns back the control.

Squeak and GNU Smalltalk already have the mechanisms for it, so I think it may be enough just to provide a dialect-independent way for C callouts based on the top of the existing provided opportunities.

The syntax described above forces the programmer to create a new method each time he wants to bind a new C function. Also, the programmer has to have a class where he (or she) will define these methods. It is not always good, so I think that the cool Alien's ability of the "in-place" C callouts should also be included into the new FFI.

5 Conclusion

This is my understanding of the task, and I will try to implement the project according this vision. Any your comments are warmly welcomed!

Dmitry.