Dynamic Languages - Past, Present and Future

David Chisnall

November 13, 2008
What is a language?

- A means of communication.
- Something both parties can understand - the computer and the programmer.

A good (programming) language is easy for both parties to understand.
The Spair-Whorf Hypothesis

- Postulates a causal relationship between natural language and thought patterns.
- More relevant to computer languages than natural ones:
  - Natural languages can easily steal parts from others if they aren’t expressive enough.
  - "Notation as a tool of thought" - Kenneth E. Iverson, 1980 Turing Award Lecture.

Language semantics affect the kind of programs we write.
Examples

- C++ has no GC, which encourages copying rather than aliasing.
- Tail-recursion optimisations in languages encourage recursive algorithms.

Consider widely different languages, like Prolog and Erlang—the capabilities of the language have significant impact on the structure of the code.
What is a ‘Dynamic’ Language

- The core language is very small.
- Anything outside the core language can be replaced by the user.
- If a static type system is present, it does not affect run time semantics.

In a dynamic language, customisation of the language is not just possible, but encouraged.
The Smalltalk Family

- Smalltalk - first dynamic, object-oriented, language.
- Self - Smalltalk without classes.
- JavaScript - Self with Java syntax.
A Quick Introduction to Smalltalk

Assignment:

1. `aVariable := anExpression.`

Message Sending:

1. `'Unary message'`
2. `receiver run.`
3. `'Message with one parameter'`
4. `receiver doSomethingTo: anObject.`
5. `'Message with two parameters'`
Smalltalk Flow Control

- No flow control other than message sending in the language.
- If statements implemented with *blocks* (closures).
- Block literals are defined with square brackets.

```plaintext
1 aBoolean ifTrue: [ obj message. ]
2 ifFalse: [ obj2 message. ].
```

True and False are subclasses of Boolean with different implementations.

```plaintext
1 whileTrue: body
2 ^self value ifTrue: [
3     body value.
4     self whileTrue: body
5   ].
```
Why is this important?

- User can add new flow control structures (e.g. for-each loops, map, select).
- User-defined and system-defined versions are indistinguishable.
- Encourages users to define new ones where convenient.
How does message sending work?

Example in Objective-C - set of Smalltalk-like extensions to C

1. `[aDictionary setObject:anObject forKey:aKey];`

Becomes:

1. `// This would really be cached`
2. `SEL selector = sel_get_uid("setObject:forKey");`
3. `IMP method = objc_msg_lookup(aDictionary, selector);`
4. `method(aDictionary, selector, anObject, aKey);`
Looking up a method dynamically is slower than calling a function directly.

The compiler can’t inline functions called indirectly.

The compiler can’t perform interprocedural analyses on indirect calls.
How do we make it fast?

- Cache the results of the lookup.
  - *Polymorphic Inline Caching* caches a set of (class, method) pairs at each call site.

- Heuristic optimisations (guess-and-test):
  - Use static analysis to work out what the class *might* be.
  - Optimise as if it is.
  - Add a branch in case it isn’t.
Do we actually need to?

- Often ‘fast’ and ‘flexible’ are not needed in the same bit of code.
- Why write them both in the same language?
The Pragmatic Smalltalk Compiler

- Initial release as part of Étoilé 0.4.
- ABI-compatible with Objective-C.
- JIT compiler based on LLVM.
  - Also does static compilation (as of Monday).
  - Produces .o files that can be linked with C/ObjC code.
What does this mean?

- Objective-C is a pure superset of C.
  - Calling C from Objective-C has no overhead.
  - C or Objective-C can have inline assembly (if you really need it).
- Objective-C objects and Smalltalk objects use the same structure.
- An object can have methods in Objective-C and Smalltalk.
- No virtual machine, just a small runtime library.
Can we make it even more flexible?

- `class \times selector \rightarrow method` - class-based dispatch.
- `object \times selector \rightarrow method` - prototype-based dispatch.
- `sender \times object \times selector \times type \rightarrow type \times context \times method \times object` - Étoilé runtime’s multidimensional dispatch.

Context Oriented Programming and other buzzwords use multidimensional dispatch.
What can we do with this?

- Change behaviour based on sender, e.g. enforce runtime visibility constraints.
- Use the *type* modification for interaction between typed and untyped languages.
- Embed arbitrary information with the context.
And that’s useful because...?

We can...

- …track messages going in to a group of objects, record them, and log the entire history of an object.
- …isolate less-trusted portions of a program.
- …track messages travelling between threads for safe concurrency.

Real example:

```objc
MKMusicPlayer *player = [[MKMusicPlayer alloc] initWithDefaultDevice];
// Move the player into a new thread.
player = [player inNewThread];
```

All messages sent to `player` are sent to the other thread, all replies are futures.
Old performance data - Étoilé runtime gives much better performance with the new sparse array implementation (equivalent to GNU).
Object Planes

- Associate a grouping of objects with a mediator object.
- Intercept messages flowing between planes.

Work conducted in collaboration with Dr. Damien Pollet at INRIA Lille.
Dynamic languages encourage loose-coupling, which makes code less fragile.

They make reasoning about code harder (but we’re working on that).

They make programming more fun!

Obligatory Étoilé plug:

http://etoileos.com