Extracted from:

The dRuby Book

Distributed and Parallel Computing with Ruby

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

The dRuby Book

Distributed and Parallel Computing with Ruby

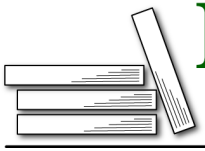
Masatoshi Seki

Translated by Makoto Inoue

Foreword by Yukihiro “Matz” Matsumoto

Edited by Susannah Davidson Pfalzer

The Facets of Ruby Series



Pragmatic Bookshelf

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *http://pragprog.com*.

The team that produced this book includes:

Susannah Pfalzer (editor)
Potomac Indexing, LLC (indexer)
Kim Wimpsett (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Original Japanese edition:
"dRuby niyoru Bunsan Web Programming" by Masatoshi Seki

Printed in the United States of America.
ISBN-13: 978-1-934356-93-7
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—March 2012

2.2 Design Principles of dRuby

I designed dRuby to extend Ruby method invocation over networks. dRuby is a library to implement distributed objects in Ruby.

dRuby has the following characteristics:

- Limited to Ruby
- 100 percent written in Ruby
- No IDL required

Let's look at these concepts a little more closely.

Pure Ruby

dRuby is a distributed object system purely targeted to Ruby. It sounds limiting, but this also means you can run dRuby in any environment where Ruby can run (see Figure 10, *The software layers*, on page 6). This is very similar to Java RMI, which also can run anywhere Java can run.

Figure 11, *An example of a system across multiple OSs*, on page 6 shows the architecture of a complex system across different operating systems.

dRuby is written purely in Ruby without using any C extension libraries—another bonus. Ruby comes with network, thread, and marshaling-related libraries as part of its standard library, so I was able to write everything in Ruby. The first version of dRuby had only 160 lines (the current dRuby has more than 1,700 lines including RDoc), and the core part of the library is still the same (see *The First dRuby*, on page 7). The concise codebase of dRuby demonstrates how easily you can write a complex library by using just Ruby's standard libraries.

Feels Like Ruby

I paid special attention to the compatibility between dRuby and Ruby. Many features of Ruby remain in dRuby, too.

Ruby is very dynamic. You don't need to use inheritance most of the time because the variables of Ruby aren't typed. Ruby looks up methods at execution time (method invocation time); these characteristics also apply to dRuby. dRuby doesn't have type-in variables, and method searches are done at execution time. Because you don't need to prepare the list of methods and their inheritance information, you don't need to write IDL.

dRuby's core mission isn't about changing the behavior of Ruby, apart from extending Ruby method invocation across networks. With this functionality,

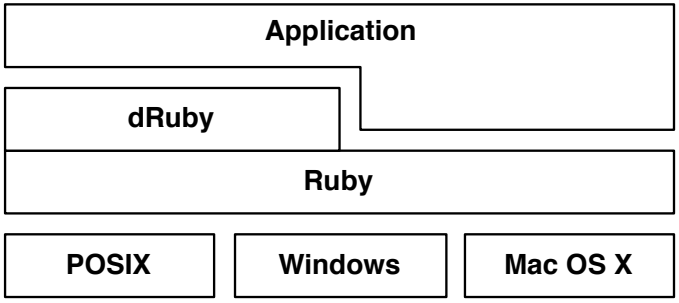


Figure 10—The software layers. Observe where dRuby sits above the operating systems.

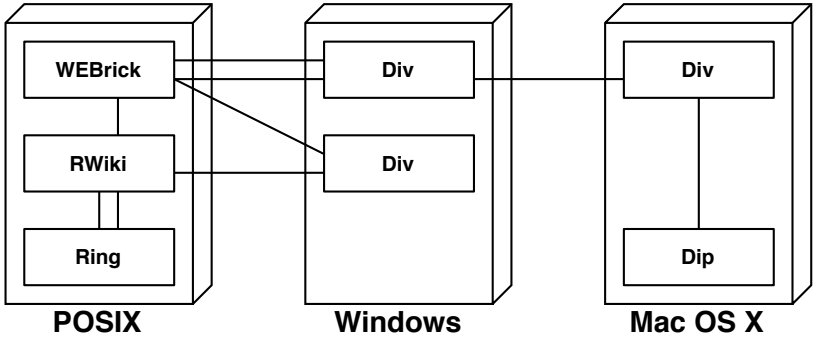


Figure 11—An example of a system across multiple OSs. Div, Ring, Dip, and RWiki are applications that use dRuby.

you can have as much ease and fun programming in dRuby as you do with Ruby. For example, you can still use a block for method calls and use exceptions as well. Other multithreading synchronization methods, such as Mutex and Queue, are also available remotely, and you can use them to synchronize multiple processes.

Pass by Reference, Pass by Value

Having said all that, sometimes you need to know the difference between Ruby and dRuby.

In Ruby, objects are all exchanged by reference when you pass or receive method arguments, return values, or exceptions.

The First dRuby

The first version of dRuby was created in 1999 and posted to the Japanese Ruby user mailing list.[a] The email is written in Japanese, but you can see some snippets of source code that look very similar to dRuby now.

```
# Starting drb server.
DRb.start_server('druby://hostname:port', front)
# Connecting to the remote server
ro = DRbObject.new(nil, 'druby://server:port')
ro.sample(1, DRbEx.new(2), 3)
```

The original source code is about 160 lines, and about 50 lines of the core code will give you clear idea about how dRuby works internally.

```
class DRbObject
  def initialize(obj, uri=nil)
    @uri = uri || DRb.uri
    @ref = obj.id if obj
  end
  def method_missing(msg_id, *a)
    succ, result = DRbConn.new(@uri).send_message(self, msg_id, *a)
    raise result if ! succ
    result
  end

  attr :ref
end
```

DRbObject acts as a proxy object, so it doesn't have any methods. Therefore, method_missing receives all the method calls and sends them to the DRbConn class.

```
class DRbConn
  include DRbProtocol

  def initialize(remote_uri)
    @host, @port = parse_uri(remote_uri)
  end
  def send_message(ref, msg_id, *arg)
    begin
      soc = TCPSocket.open(@host, @port)
      send_request(soc, ref, msg_id, *arg)
      recv_reply(soc)
    ensure
      soc.close if soc
    end
  end
end
```

DRbConn acts as a TCPSocket server. It transfers the message to the remote server—so simple. If you're interested in the internals of dRuby, read the rest of the original code to get a better idea about the structure of the library before jumping into reading the current version of dRuby, which is more than 1,700 lines.

a. http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-list/15406

In dRuby, objects are exchanged either by reference or by the copy of the original value. When an object is passed by reference, the operation to the object will affect the original (like a normal Ruby object). However, when passed by value, the operation doesn't affect the original, and the change stays within the process where the object is modified (see Figure 12, *Passing by reference and passing by value*, on page 9).

This difference doesn't exist in Ruby, and you have to pay special attention to this when you program with dRuby.

If I really wanted to make dRuby look the same as Ruby, I could have designed dRuby to always pass by reference. However, remote object method invocation requires some network overhead, and doing small object operations all via RMI isn't effective. It's vital to pass the copy of the object in certain situations to increase your application's performance. Also, if you keep passing by reference, you'll never get the actual value from the remote server. Instead, you'll be looping forever, trying to find out the object's state.

It might sound a bit complicated, but don't worry. You don't have to specify whether you plan to pass by value or by reference—dRuby does it for you. Because dRuby automatically selects the reference method, you don't have to write a special method for dRuby.

You'll see more detail about automatically passing by value or by reference in Chapter 4, *Pass by Reference, Pass by Value*, on page ?.

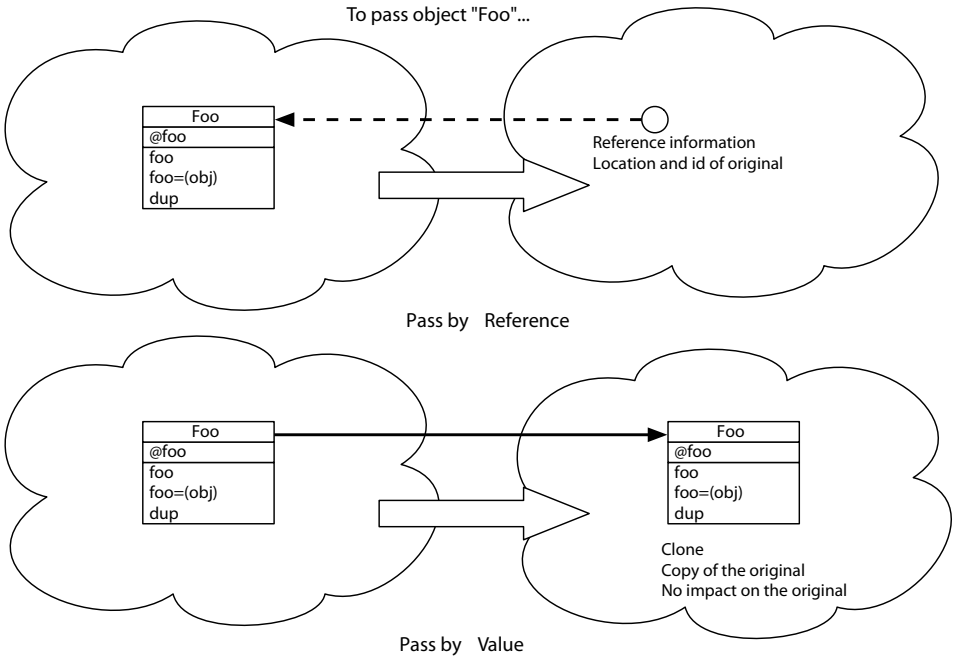


Figure 12—Passing by reference and passing by value