CPSC 551 - Distributed Systems - Fall 2019

Project 1, due October 14

Background

Note 2.3 on pp. 68-70 of the textbook discusses Linda tuple spaces and provides example code using a Python Package named *PyLinda*. Unfortunately,

- The PyLinda package is not not the same as the pylinda package available on PyPI.
- PyLinda was written for Python 2.
- The available code for PyLinda is missing some source files.
- The original author of the code seems to have disappeared from the Internet.

Fortunately Ruby, another common scripting language, includes a module named *Rinda* which implements the Linda distributed computing paradigm. Even better, Chapter 2 of the textbook describes the *adapter* pattern for middleware, so that we can bridge the gap between Python and Ruby.

Starting with Rinda

Write a Ruby server program `tuplespace.rb` to create a Rinda TupleSpace and begin listening on a TCP port for incoming dRuby connections.

Write a Ruby client program `blog.rb` to connect to the Rinda TupleSpace and perform the same operations as the example code fragments in Note 2.3 in the textbook. (Alternatively, you might implement these operations as three separate client programs `bob.rb`, `alice.rb`, and `chuck.rb`.)

Note: the Ruby equivalent of the Python `str` type is `String`, and the `in(t)`, `rd(t)`, and `out(t)` operations are named `take(t)`, `read(t)`, and `write(t)`, respectively.

Adapting Rinda to XML-RPC

According to WIkipedia, "XML-RPC is a remote procedure call (RPC) protocol which uses XML to encode its calls and HTTP as a transport mechanism." XML-RPC support is included in the Python Standard Library and until recently was included in the Ruby Standard LIbrary as well.

Write a Ruby server program `adapter.rb` to act as an XML-RPC server. Connect to the Rinda TupleSpace with a `Rinda::TupleSpaceProxy` and add XML-RPC handlers named `'_in'`, `'_rd'`, and `'_out'` to allow the TupleSpace operations to be called via XML-RPC.

Adapting Rinda templates

In the PyLinda calls to `_rd()` shown in Note 2.3, you'll note that some tuple fields can be literal strings (e.g., `"bob"` or `"distsys"`), while some are Python types such as `str`. These Python types are used as templates, matching any value of the given type in the specified tuple field.

Rinda includes a similar feature, with a variety of template types. In addition to literals for the standard XML-RPC primitive types, you will need to support the following Rinda template options:

Rinda template type	Python equivalent for XML-RPC calls	Matches	Ruby constructor method
`nil`	`None`	Any object	*(No constructor needed)*

Ruby classes
 	
e.g. `String`, `Numeric`
	`{ 'class': `*`className`*` }`
	
	e.g.
	`{ 'class': 'String' },`
	`{ 'class': 'Numeric' }`
		Objects of the given class	`Module.const_get(`
			`  `*`className`*
			`)`

Regular expressions
 	
e.g. `%r{^[-+/*]$}`
	`{ 'regexp': `*`regExp`*` }`
	
	e.g.
	`{ 'regexp': '^[-+/*]$' }`
		Strings matching the given regular expression	`Regexp.new(`*`regExp`*`)`

Ranges
 	
e.g. `1..10`, `'a'..'z'`
	`{ 'from': `*`start`*`, 'to': `*`end`*` }`
	
	e.g.
	`{ 'from': 1, 'to': 10 },`
	`{ 'from': 'a', 'to': 'z' }`
		Values in the given interval	`Range.new(`
			`  `*`start, end`*
			`)`

Symbols
 	
e.g. `:chopstick`
	`{ 'symbol': `*`name`*` }`
	
	e.g.
	`{ 'symbol': 'chopstick' }`
		The same symbol. When used as the first item in a tuple, symbol matching is optimized for fast searching.	*`name`*`.to_sym()`

Examples

The PyLinda call

```
t1 = blog._rd(("bob", "distsys", str))
```

Might translate into the following Python XML-RPC call:

```
t1 = blog._rd(['bob', ‘distsys', { 'class': 'String' }])
```

while the Rinda call

```
ops, a, b = ts.take([ %r{^[-+/*]$}, Numeric, Numeric])
```

Might translate into the following Python XML-RPC call:

```
ops, a, b = ts._in([
    { 'regexp': '^[-+/*]$' },
    { 'class': 'Numeric' },
    { 'class': 'Numeric' }
])
```

Using Rinda from Python

Write a Python client program `blog.py` to connect to the XML-RPC adapter and perform the same operations as the example code fragments in Note 2.3 of the textbook. (Alternatively, you might implement these operations as three separate client programs `bob.py`, `alice.py`, and `chuck.py`.)

Write Python client programs `arithmetic_server.py` and `arithmetic_client.py` equivalent to the example programs for performing arithmetic operations shown on the Wikipedia page for Rinda.

Finally, solve the “exercise to the reader” described at the end of Note 2.3 by writing a Python client program `next.py` that extends the code fragments for Bob, Alice, and Chuck such that a *next* message will be selected instead of a random one.

Platform

You may use any platform to develop and test the project, but note that per the Syllabus the test environment for projects in this course is a Tuffix 2019 Edition r2 Virtual Machine with Python 3.6.8 and Ruby 2.5.1p57. It is your team’s responsibility to ensure that your code runs on this platform.

Libraries

The Python xmlrpc.client and Ruby Rinda modules are included in the language’s standard libraries. XML-RPC support was removed from the Ruby standard library starting in version 2.4, but can be installed as a Ruby Gem (similar to a Python Package). To install the `xmlrpc` gem on Tuffix, use the following command:

```
$ gem install --user-install xmlrpc
```

Tips

- For a quick tutorial on Ruby, see Ruby in Twenty Minutes.
- For differences between Python and Ruby, see To Ruby From Python.
- See https://github.com/ProfAvery/cpsc551/tree/master/xmlrpc for some simple XML-RPC clients and servers.
- To allow Python clients to pass `None` to XML-RPC functions, call `xmlrpc.client.ServerProxy(uri, allow_none=True)`
- To allow `nil` to be passed to and from Ruby XML-RPC servers, see `allow_nil.rb`.
- The Ruby `inspect` method can be useful for examining parameter values.
- Ruby blocks do not use explicit `return` statements. Instead, they return the last expression evaluated. XML-RPC may or may not be able to marshal the value of that expression. You may see an error like `ERROR RuntimeError: Wrong type!`
- Note that `{ 'symbol':` *name* `}` is both an input and an output format (i.e., `_in()` and `_rd()` can return symbols as well as matching them).

Submission

Submit your project by uploading your Python and Ruby source code and any other relevant artifacts to the `project1/` subdirectory of the folder that will be shared with you on Dropbox.

You may work alone, or make a single submission for a team of 2-3 students. If you work in a team, only one submission is required, but for safety consider uploading copies to each team member's submission folder. (Make certain, however, that the copies are the same in each case; the instructor will not attempt to ascertain which is the “real” submission.)

A printed submission sheet will be provided on the due date. To finalize your submission, fill out the sheet with the requested information and hand it in to the professor by the end of class. Failure to follow any submission instructions exactly will incur a **10%** penalty on the project grade for all team members.