Ship – A little logistics language

Final Report – COMS 4115

Kamil Alexander
ak2834@columbia.edu

1. Introduction

Ship is a simple interpreter that reads a configuration file and spits out Ruby objects.

It is intended for automation of data processing tasks in fields such as bioinformatics where the inputs consist of multiple data sources and the execution pipeline is comprised of a series of independent steps. Each step usually involves running a command-line tool with multiple configuration parameters and manipulating the inputs and outputs with UNIX shell commands and ad-hoc scripts. With large data volumes it is sometimes necessary to process the data on multiple machines in parallel which makes job coordination and data logistics time-consuming and error-prone.

Ship simplifies task orchestration by freezing the job settings in a Shipfile spec, similar to Makefile [1] used in software build and deployment process, and with a similar motivation, i.e. being able to combine the commands for the different targets into a single file and being able to abstract out dependency tracking and data handling. Ship translates the commands provided in Shipfile into shell commands suitable for local or remote execution. It makes data-driven experiments more repeatable and controlled as it minimizes the room for error inherent in typing command line options by hand.

The parser was developed with ANTLR3 for Ruby [2], the grammar is partially based on Fig Configuration Language Interpreter by Terrence Parr [3]. Ship interprets input file (Shipfile) into Ruby objects, which allows easy integration within Ruby ecosystem (e.g. it can be plugged in to command-line shell such as rush [4]).

2. Code Example

```
//Shipfile
Source s1 {  input_dir='~/data'; files ='*.in'; hosts = $hosts; modified_on ='*';}
Source s2 {  input_table =pcbl_s.batch.references; db= $bdb; hosts =$hosts;}
Target t1 {  output_dir='~/results';  files ='*.dat'; hosts =$hosts;}
Target t2 {  output_dir='~/summary';  files ='*.txt'; hosts ='localhost';}
Target t3 {  final_dir='~/experiments/summaries'; hosts =$hosts;}
TASK1{
            input1 =select * from $s1 where pattern= '^ref\.[0-9]+';
            input2 = select * from $s2 where CF2= '77.890' and GNFR = 'Y';
            map = mapper $opts $input1 $input2 $t1;
            shuffle = shuffler $t1;
}
TASK2 {reduce = reducer $t1, $t2;}
TASK3 {send_command=Send $t2 to $t3, Notify on Timeout;}
ALL {pipeline= [$TASK1, $TASK2, $TASK3];}
CLEAN {remove = [$t1.files, $t2.files];}

options opts {
      -c = 2000;
       -l = ['10.1.1.190', '10.1.1.191'];
}
database bdb {
      port = 80;
      user = "test";
      rootdn= "cn=Manager,dc=example,dc=com";
}
hosts = {
            h=[algiers.clic.cs.columbia.edu,
            ankara.clic.cs.columbia.edu,
            baghdad.clic.cs.columbia.edu];
}
```

3. Lexical Conventions

Ship supports the following kinds of tokens: identifiers, strings, integers, expressions.
In general blanks, tabs, newlines are ignored except as they serve to separate tokens.

Identifiers
```
ID  :   ('_'|'-'|'a'..'z'|'A'..'Z') ('_'|'-'|'\.'|'a'..'z'|'A'..'Z'|'0'..'9')* ;
```

Strings
```
STRING :('\".*\"|"".*"");
```

Integers
```
INT :   '0'..'9'+ ;
```

Whitespaces
```
WS  :   (' '|'\n'|'\t')+;
```

Comments
```
CMT :   '/*' .* '*/';
```

4. Syntax

Shipfile can contain several *object* declarations, where each object has a label (*qid*) and a block declaration enclosed with curly braces. The block contains *assign* statements in the form *ID=expression*; Expressions can be terminal *strings*, *integers*, *identifiers*, or non-terminal *queries*, *commands* and *lists of expressions*.

file
```
   : object+
   ;
```
object
```
  :      qid ID? '{' assign* '}'
   ;
```
qid
```
  :   ID ('.' ID)*
  ;
```
assign
```
   :   ID '=' expr ';'
   ;
```
expr
```
   :   STRING
   |   ('$')?ID
   |   INT
   |   '[' ']'
   |   query
   |   command
   |   '[' e=expr(',' e=expr )* ']'
```

Commands

Ship currently supports two built-in commands:

1. *Select* query which can be applied to filtering and extraction from two types of data sources:
    a. unstructured data, such as text files by searching for a *pattern*, which can be a regular expression
    b. structured data, such as database tables, where we can filter using *where* clause criteria
2. *Send* command which allows to ship files to target destination (i.e. remote host). Delivery options such as notification mode and transfer type can be optionally added by the user.

Ship grammar can be easily extended to include domain-specific commands.

Query statement

```
query
    :   select_stmt where_stmt ;
select_stmt
    :  'select' '*' 'from' (directory | table);
where_stmt
    :  ('where' clause ('and' clause)*) ?;
clause
    : file_name
    | pattern
    | ID '=' STRING ;
directory
      : '/' ID;
table
      : ID;
file_name
      : 'file'  '=' STRING;
pattern
      :   'pattern'  '=' STRING;
```

SendCommand statement

```
command
     :  (send_command delivery_options*
     |  wait_command)
     ;
send_command
     :              'Send' container? 'to' destination ('by' send_channel)?;
container:  ID;
destination: ID     ;
send_channel: ID;
delivery_options
    : 'Get' 'Signature'
    |  notify_on;
notify_on
     : 'Notify' notify_recipient 'on' notify_criteria;
notify_recipient returns [value]
     : SENDER
     | ALL;
notify_criteria returns [value]
      : SUCCESS
      | FAILURE
      | TIMEOUT;
wait_command:      'wait' 'for' wait_criteria  notify_on?;
wait_criteria: ID;
```

5. ANTLR Grammar (without Action blocks)

```
grammar Ship_plain;

tokens {
  ALL        = 'All';
   SENDER   = 'Sender';
   SUCCESS      = 'Success';
   FAILURE       = 'Failure';
   TIMEOUT         = 'Timeout';
 }
file : object+ ;
object
  :       qid ID? '{' assign* '}'
   ;
assign
   :   ID '=' expr ';'
   ;
expr:   STRING
   |   INT
   |   '$' ID
   |   '[' ']'
   |   query
   |   command
   |   '[' expr (',' expr)* ']'
   ;
qid :   ID ('.' ID)* ;
query
    :    select_stmt where_stmt
    ;
select_stmt
    :  'select' '*' 'from' (directory | table)
    ;
directory
      : '/' ID
      ;
table
      : ID
      ;
where_stmt
    :  ('where' clause ('and' clause)*) ?
    ;
clause
    : file_name
    | pattern
    | ID '=' STRING
    ;
file_name
      : 'file'  '=' STRING      ;
pattern
      :   'pattern'  '=' STRING      ;
command
     :  (send_command delivery_options*
     |  wait_command)
     ;
send_command
     : 'Send' container 'to' destination 'by' send_channel
     ;
container
     :  ID
     ;
destination
     : ID
     ;
send_channel
     : ID
     ;
delivery_options
    : 'Get' 'Signature'
    |  notify_on
```

```
   ;
notify_on
    : 'Notify' notification_recipient 'on' notification_criteria
    ;
notification_recipient
    : SENDER
    | ALL
    ;
notification_criteria
     : SUCCESS
     | FAILURE
     | TIMEOUT
     ;
wait_command
      :      'wait' 'for' wait_criteria  notify_on?
      ;
wait_criteria
      : ID
      ;

STRING :('\".*'\"|'".*'");
INT :   '0'..'9'+ ;
ID  :   ('_'|'-'|'a'..'z'|'A'..'Z') ('_'|'a'..'z'|'A'..'Z'|'0'..'9')* ;
WS  :   (' '|'\n'|'\t')+ {$channel=HIDDEN;} ;
CMT :   '/*' .* '*/'     {$channel=HIDDEN;} ;
```

6. ANTLR Grammar (with Ruby Action blocks)

```
grammar Ship;

tokens {


    ALL        = 'All';
    SENDER   = 'Sender';
    SUCCESS      = 'Success';
    FAILURE       = 'Failure';
    TIMEOUT         = 'Timeout';
  }

@init {
}
@members {

class CMD
  class << self; attr_accessor :destination,:signature,:notify_on,:notify_recipient,:notify_criteria end
  @destination='localhost';
  @signature='No' ;
  @notify_on ="Yes"
  @notify_recipient='All';
  @notify_criteria='Success';
 def self.toString
                    cmd_str= "send -host=" << destination << " -signature=" << signature << " -notify_on=" << notify_on << " -notify_recipient=" <<
notify_recipient << " -notify_criteria=" << notify_criteria                    ;
                    puts cmd_str;
 end
end

class Query
  class << self; attr_accessor :connection_string,:query end
  @query='';
  @connection_string='default' ;
 def self.toString
       query_str= "sql -connection_string=" << connection_string << " -query= " <<query ;
       puts query_str;
 end
end

}
file
    : object+ {CMD.toString;} {Query.toString;}
    ;
object
   :       qid {#puts $qid.text;} (ID {#puts $ID.text;})? '{' assign* '}'
    ;
assign
    :   ID '=' expr ';' {#puts $ID.text;} {#puts $expr.value.to_s;}
    ;
expr returns [value]
    :   STRING {$value= $STRING.text;}
    |   ('$')?ID {$value= $ID.text;}
    |   INT {$value= $INT.text;}
    |   '[' ']'
    |   query {$value=$query.text;}{Query.query = $value}
    |   command {$value=$command.text;}
    |    {elements= Array.new;}
       '[' e=expr {elements.push($e.value);}
                    (',' e=expr {elements.push($e.value);} )* ']'
       {$value=elements;}
    ;
qid
   :   a=ID ('.' b=ID {#puts $b.text;})*  {#puts $a.text;}
   ;
query
     :    select_stmt where_stmt
     ;
```

```
select_stmt
    :  'select' '*' 'from' (directory | table)
    ;
directory
     : '/' ID
     ;
table
     : ID
     ;
where_stmt
    :  ('where' clause ('and' clause)*) ?
    ;
clause
    : file_name
    | pattern
    | ID '=' STRING
    ;
file_name
     : 'file'  '=' STRING
     ;
pattern
     :   'pattern'  '=' STRING
     ;

command
     :  (send_command delivery_options*
     |  wait_command)
     ;
send_command
     :              'Send' container? 'to' destination ('by' send_channel)?
     ;
container
     :  ID
     ;
destination
     : ID {CMD.destination=$ID.text;}
     ;
send_channel
     : ID
     ;
delivery_options
    : 'Get' 'Signature' {CMD.signature="Yes";}
    |  notify_on  {CMD.notify_on="Yes";}
    ;
notify_on
     : 'Notify' notify_recipient 'on' notify_criteria
       {CMD.notify_recipient=$notify_recipient.value.to_s;}
                 {CMD.notify_criteria=$notify_criteria.value.to_s;}
     ;
notify_recipient returns [value]
     : SENDER {$value=$SENDER;}
     | ALL {$value=$ALL;}
     ;
notify_criteria returns [value]
     : SUCCESS {$value=$SUCCESS;}
     | FAILURE {$value=$FAILURE;}
     | TIMEOUT {$value=$TIMEOUT;}
     ;
wait_command
     :      'wait' 'for' wait_criteria  notify_on?
     ;
wait_criteria
     : ID
     ;

STRING :('\".*\"|"".*"");
INT :   '0'..'9'+ ;
ID  :   ('_'|'-'|'a'..'z'|'A'..'Z') ('_'|'-'|'\.'|'a'..'z'|'A'..'Z'|'0'..'9')* ;
WS  :   (' '|'\n'|'\t')+ {$channel=HIDDEN;} ;
CMT :   '/*' .* '*/'     {$channel=HIDDEN;} ;
```

7. Driver program (Ship.rb)

```
#Ship.rb
#!/usr/bin/env ruby
require 'rubygems'
require 'antlr3'
require 'ShipLexer.rb'
require 'ShipParser.rb'
if ARGV.size==0
puts "Missing input file"
exit
end
lexer = open( ARGV[0] ) do | f | Ship::Lexer.new( f ) end
tokens = ANTLR3::CommonTokenStream.new( lexer )
parser = Ship::Parser.new( tokens )
parser.file
```

8. Test input file (input.conf)

```
Package P1 {
        query=select * from Fruits where name= 'Jincheng' and location = 'Jiangxi' ;
}
Command C1{
        command= Send P1 to Lisbon by Air Get Signature Notify All on Success;
}
database bdb {
        port = 80;
        user = "test";
        suffix= "dc=example,dc=com";
        rootdn= "cn=Manager,dc=example,dc=com";
        portal = "www.giuou.com";
        urls = ["http://test1.com", "www.test2.com"];
}
options opts {
        -u = nobody;
        -c = 2000;
        -port = 3137;
        -l = ['10.1.1.190', '10.1.1.191'];
        -logfile= '/var/log/ship.log';
        -set-max-intset-entries = 1;
}
Client {
        ids = [$bdb, $opts];
}
```

9. Installation Instructions

Prerequisites: Ruby, Ruby Gems, ANTLR3 gem, git
git clone https://github.com/akamil/ship/
cd ship
antlr4ruby Ship.g
ruby Ship.rb input.conf

References

1. GNU Make: http://www.gnu.org/software/make/
2. ANTLR3 for Ruby: http://antlr.ohboyohboyohboy.org/
3. Fig Generic Configuration Language Interpreter: http://www.antlr.org/wiki/display/ANTLR3/Fig+-+Generic+configuration+language+interpreter
4. Rush, a Ruby replacement for the UNIX shell: http://rush.heroku.com/