Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 15, 2007

S. Bortzmeyer
AFNIC

November 11, 2006

Cosmogol: a language to describe finite state machines
draft-bortzmeyer-language-state-machines-01

Status of this Memo

Copyright Notice

Abstract

Several RFCs contain a state machine to describe a protocol. There is no standard way of describing such a machine, the most common way being an ASCII-art diagram. This document specifies an other solution: a domain-specific language for finite state machines. It allows state machine descriptions to be automatically checked and may be translated into other formats. Its purpose is to provide a stable reference for RFCs which use this mini-language.

Table of Contents

1. Requirements notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [1].

2. Introduction

One can find finite state machines, for instance, in RFC 793 [3] or RFC 4340 [8]. The Guide for Internet Standards Writers [5], in 2.12 "Notational conventions" and 3.3 "State machines description", lists several ways to describe them but does not recommend one. Unlike grammars, which are always specified with ABNF [2], state machines have no standard description language. RFCs typically use figures, list of transitions or tables.

Figures (wether in ASCII-art, in Unicode-art, in SVG, in GIF or whatever) are:

- impossible to analyze automatically (for instance to check if they are deterministic),

- not readable if the state machine is large.

Another issue, and one which created a lot of discussions, is the "need" to allow something more than US-ASCII (and some people require even more than raw text) in the RFCs. A common "use case" is this need to specify state machines through drawings. That it is not the only way and not even the best way and the choice here is to use an ASCII-based languages, thus requiring no change in the format of the RFC.

Informal natural language text is not perfect either, because it impossible to analyze automatically (for instance to check if they are complete).

Tables are also a possible solution (if the machine is finite). They are fine for automatic processing but very bad for presentation to humans, specially if they are large. Most people find them too low-level.

To conclude, let us note that RFC 4006 [7] uses a list of tuples, each tuple being a transition. Although the (informal) syntax it uses is not parsable by a program, the idea behind it is close from the Cosmogol language.

3. Terminology

TODO: because of the state of this document, some choices are not final. Every time you see the word ALTERNATIVE in uppercase, it means several possible choices are listed.

The terminology of state machines is not perfectly standard. We use here the words:

- state,
- message, the condition of a transition,
- action, performed after the transition.

The Cosmogol language contains declarations, assignments and transitions. A declaration announces that a name will be used for either a message, a state or an action. An assignment binds a value to a variable.

A transition is described by the name of the message, the names of the current and next state and an optional action. They are the heart of the Cosmogol language: in Cosmogol, a state machine is a list of transitions.

A processor is a program that processes Cosmogol files. It can be validating or not. Any processor MUST check the syntax of the file. A validating processor MUST perform the checks described in Section 5.

In addition to the checks, a processor MAY perform other tasks such as translating to another format, for instance Graphviz [9].

TODO: some way to modularize state machines? For instance, X509 checking is described by several SM.

4. Grammar

Here is the grammar of Cosmogol, using ABNF [2]

```
state-machine = 1*(statement / (*comment-wsp))

statement = (declaration / transition / assignment)
            *comment-wsp ";" *comment-wsp

colon = *comment-wsp ":" *comment-wsp
comma = *comment-wsp "," *comment-wsp
equal = *comment-wsp "=" *comment-wsp
arrow = *comment-wsp "->" *comment-wsp

declaration = names colon value
; ALTERNATIVE: indicate the possible values in the grammar:
; declaration = names colon type
; type = "state" / "message" / "action"

assignment = name equal value

names = name *(comma name)

name = quoted-name / regular-identifier

quoted-name = DQUOTE 1*(identifier-chars) DQUOTE

; TODO: this grammar allows identifiers like foo----bar
; (several dashes). Do we really want it?
regular-identifier =
            ALPHA /
            (ALPHA *(ALPHA / DIGIT / "-") (ALPHA / DIGIT))

transition = current-states colon
             messages arrow next-state
             [colon action]

; ALTERNATIVE : some people prefer to put the message first:
;transition = message colon
;             current-state arrow next-state
;             [colon action]

; ALTERNATIVE: some people prefer to see the current-state and
; the message grouped together:
;transition = left-paren current-state comma message right-paren
;             arrow next-state
;             [colon action]
```

```
   ; ALTERNATIVE: allow some grouping, for instance:
   ;   Signal1:
   ;     IDLE -> BUSY:
   ;       connectSubscriber;
   ;     CONNECTING -> DISCONNECTING:
   ;       disconnectSubscriber
   ; # Henk-Jan van Tuyl <hjgtuyl@chello.nl>

   ; ALTERNATIVE: allow more than one action, comma-separated
   ;  Marc Petit-Huguenin <marc@8x8.com>

   current-states = name *(comma name)
   messages = name *(comma name)
   next-state = name
   action = name

   value = regular-identifier / quoted-name

   identifier-chars = ALPHA / DIGIT /
                       "-" / "_" / "'" / "," / ";" / SP
                      ; All letters and digits and
                      ; some (a bit arbitrary) chars

   comment        =  "#" *(WSP / VCHAR) CRLF

   comment-nl           =  comment / CRLF

   comment-wsp          =  *(WSP / comment-nl)
```

5. Semantics

A validating processor MUST perform all these checks.

Every message, state and action MUST be declared. The possible values for the right side of a declaration are:

- MESSAGE
- STATE
- ACTION

The order between statements (transitions, declarations and assignments) has no meaning. For instance, the declaration of a message can takes place after its use in a transition.

All names are case-sensitive. ALTERNATIVE: make them case-insensitive, which is possible since everything is in US-ASCII.

TODO: should we document naming *conventions*, such as "States in uppercase, messages in capitalized"?

Assignments are only possible to pre-defined variables. No assignment is mandatory. The variables are:

- Title (used for some displays)
- Initial (to indicate the initial state; if this variable is assigned, every state MUST be reachable - may be indirectly - from the initial state)
- Final (to indicate the final state; if this variable is assigned, this final state MUST be reachable - may be indirectly - from every state)

When there are several current states indicated, they must be interpreted as a set. For every member of the set, the message yield to the next state. Same thing when there are several messages. This allows some grouping of similar transitions. So, the following state machine:

```
Waiting, End: timeout, user-cancel, atomic-war -> Start;
```

is to be interpreted as completely equivalent to:

```
Waiting: timeout -> Start;
End: timeout -> Start;
Waiting: user-cancel -> Start;
End: user-cancel -> Start;
Waiting: atomic-war -> Start;
End: atomic-war -> Start;
```

The state machine MUST be deterministic, that is for every couple (current state, message), there must be only one output (next state and optional action).

Besides the "Initial" variable mentioned above Paragraph 2, a processor may provide a mean to the user to declare (may be on the command line) a state as the start of the machine and the processor may check that every other state is reachable from this state, as if it were declared as "Initial". Same thing for the "Final" state.

A processor may provide a flag to require that the state machine is complete, that is every transition must be explicitely listed.

6. Internationalisation considerations

The character set of the language is US-ASCII only, for conformance with [4], section 2.1. This reflects the fact that RFC must be written in english (TODO: something which does not seem to be documented anywhere).

7. IANA Considerations

None

8. Security Considerations

Implementors of state machines are warned to pay attention to the default case, the one for which there is no explicitely listed transition.

ALTERNATIVE: force every transition to be declared. This is believed to be too demanding for large SM.

9. References

9.1. Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005.

9.2. Informative References

[3] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[4] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

[5] Scott, G., "Guide for Internet Standards Writers", BCP 22, RFC 2360, June 1998.

[6] Hollenbeck, S., "Extensible Provisioning Protocol (EPP)", RFC 3730, March 2004.

[7] Hakala, H., Mattila, L., Koskinen, J-P., Stura, M., and J. Loughney, "Diameter Credit-Control Application", RFC 4006, August 2005.

[8] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, March 2006.

[9] AT&T Research, "Graphviz, Graph Visualization Software", December 2004, <http://www.graphviz.org/>.

[10] Rapp, C., "The State Machine Compiler", January 2000, <http://smc.sourceforge.net/>.

[11] Thurston, A., "Ragel State Machine Compiler", August 2006, <http://www.cs.queensu.ca/home/thurston/ragel/>.

[12] "FSMLang", September 2006, <http://fsmlang.sourceforge.net/>.

[13] Tels, "Graph::Easy", March 2006, <http://search.cpan.org/~tels/Graph-Easy/>.

URIs

[14] <http://www.linux.com/article.pl?sid=05/11/08/2018216>

Appendix A. Examples

The TCP state machine, from RFC 793 [3].

```
# The TCP state machine. RFC 793 3.2 "Terminology"

Title = "Transmission Control Protocol";

SYN-RCVD, SYN-SENT, FIN-WAIT-1, FIN-WAIT-2, ESTAB,
      CLOSING, TIME-WAIT, CLOSED, CLOSE-WAIT,
      LISTEN, LAST-ACK : STATE;

CLOSE, passive-OPEN, active-OPEN ,
      rcv-SYN, rcv-ACK-of-FIN, rcv-ACK-of-SYN, rcv-SYN-ACK,
      rcv-FIN, SEND, Timeout : MESSAGE;

LISTEN : CLOSE -> CLOSED : Delete-TCB ;
# ALTERNATIVE syntax:
# CLOSE: LISTEN -> CLOSED : Delete-TCB ;
# ALTERNATIVE syntax:
# (LISTEN, CLOSE) -> CLOSED : Delete-TCB ;

CLOSED : passive-OPEN  -> LISTEN : Create-TCB;

LISTEN : rcv-SYN -> SYN-RCVD;

SYN-RCVD : CLOSE -> FIN-WAIT-1;

FIN-WAIT-1: rcv-ACK-of-FIN -> FIN-WAIT-2;

FIN-WAIT-1: rcv-FIN -> CLOSING;

FIN-WAIT-2 : rcv-FIN -> TIME-WAIT;

CLOSING : rcv-ACK-of-FIN -> TIME-WAIT;

CLOSED : active-OPEN -> SYN-SENT;

LISTEN : SEND -> SYN-SENT;

SYN-SENT : rcv-SYN -> SYN-RCVD;

SYN-RCVD : rcv-ACK-of-SYN -> ESTAB;

SYN-SENT : rcv-SYN-ACK-> ESTAB;

ESTAB : CLOSE -> FIN-WAIT-1;
ESTAB : rcv-FIN -> CLOSE-WAIT;

CLOSE-WAIT : CLOSE -> LAST-ACK;

TIME-WAIT : Timeout -> CLOSED;

LAST-ACK : rcv-ACK-of-FIN -> CLOSED;
```

The EPP state machine, from RFC 3730 [6].

```
# Extensible Provisioning Protocol (EPP)

"Waiting for client", "Prepare greeting", "End session",
"Waiting for client authentication", "Processing login",
"Prepare fail response", "Prepare response",
"Waiting for command", "Processing command": STATE;

"Connected or hello", "Close connection or idle",
"Send greeting", "login received", "Send response",
Timeout, "Auth fail",
"Auth OK", "Command received",
"Command processed", "Send X5xx response",
"Send 2501 response": MESSAGE;

Initial = "Waiting for client";

 "Waiting for client": "Connected or hello" -> "Prepare greeting";

 "End session" : "Close connection or idle" ->
              "Waiting for client";

"Prepare greeting": "Send greeting"->
             "Waiting for client authentication";

"Waiting for client authentication":Timeout :  -> "End session";

"Waiting for client authentication" : "login received"->
               "Processing login";

 "Processing login": "Auth fail"  -> "Prepare fail response";

"Prepare fail response":"Send response" ->
"Waiting for client authentication";

"Processing login": "Auth OK" -> "Waiting for command";

"Waiting for command": Timeout -> "End session";
```

```
"Prepare response" : "Send response" -> "Waiting for command";

 "Processing command" : "Command processed" ->
                "Prepare response";

"Waiting for command" : "Command received"  ->
                "Processing command";

"Prepare response" : "Send X5xx response" -> "End session";

"Prepare fail response" : "Send 2501 response"->
                "End session";
```

The DCCP state machine, from RFC 4340 [8].

```
# RFC 4340, 8.4. "DCCP State Diagram"

CLOSED, LISTEN, REQUEST, RESPOND, OPEN, PARTOPEN, CLOSING, TIMEWAIT,
CLOSEREQ : STATE;

Passive-open, Active-open, Receive-ack, Receive-reset,
Server-active-close, Active-close, Receive-packet,
Receive-response,
Receive-request, Timer-expires, Receive-close : MESSAGE;

CLOSED : Passive-open ->LISTEN;

LISTEN : Receive-request ->RESPOND;

RESPOND: Receive-ack ->OPEN;

CLOSING : Receive-reset ->TIMEWAIT;

OPEN : Server-active-close->CLOSEREQ;

CLOSEREQ : Receive-close ->CLOSED;

OPEN : Active-close  ->CLOSING;

REQUEST : Receive-response->PARTOPEN;

PARTOPEN : Receive-packet ->OPEN;

 CLOSED : Active-open->REQUEST;

 TIMEWAIT : Timer-expires -> CLOSED;

 OPEN: Receive-close ->CLOSED;
```

Appendix B. First implementation

The first implementation of the Cosmogol language can be found at <http://www.cosmogol.fr/>. It is a processor which is able to check state machines specified in Cosmogol and to translate them into Graphviz.

Appendix C. Related work

All of them are interesting back-ends for a Cosmogol processor:

- Graphviz [9] is a widely-used language to describe graphs. It has been used for state machines such as TCP [14]. But it is more presentation-oriented, you cannot restrict it to just the description. Consequently, there are currently no tools to check, for instance the determinism.

- The Perl module Graph::Easy [13] shares most of the aims of Graphviz. It is also oriented towards presentation.

- SMC [10], Ragel [11] and FSMlang [12] are more oriented towards code-generation.

Appendix D. Changes

D.1. Changes from -00

- The syntax of a transition is different: the current-state is now the first item, and not the message. There was a clear consensus among the reviewers on this change.

- Several messages are now allowed in a transition, to indicate a set of messages. Same thing for the current state.

- Several bug fixes in the grammar.

Appendix E. Acknowledgements

Significant contributions have been made by Pierre Beyssac, Emmanuel Chantreau, Frank Ellermann, Kim Minh Kaplan, Thomas Quinot, Bertrand Petit, Phil Regnauld, Mohsen Souissi and Olivier Ricou.

Author's Address

Stephane Bortzmeyer
AFNIC
Immeuble International
Saint-Quentin-en-Yvelines 78181
France

Phone: +33 1 39 30 83 46
Email: bortzmeyer+ietf@nic.fr
URI: http://www.afnic.fr/

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).