net2o: Command Language

A universal structured data language

Bernd Paysan

September 26, EuroForth 2014, Palma di Mallorca

Overview

Motivation

Object Oriented Forth Code as Data

A Few Examples

Forth–Style Communication

Requirements for secure communication (secure as in “no exploitation through misinterpretation”)

- Extremely simple interpreter
- Extensible, but extensions must be allowed by the receiver
- Universal, i.e. only one interpreter to audit and verify
- Triviality makes it difficult to explain

Forth–Style Communication

Requirements for secure communication (secure as in "no exploitation through misinterpretation")

- Extremely simple interpreter
- Extensible, but extensions must be allowed by the receiver
- Universal, i.e. only one interpreter to audit and verify
- Triviality makes it difficult to explain

Forth–Style Communication

Requirements for secure communication (secure as in “no exploitation through misinterpretation”)

- Extremely simple interpreter
- Extensible, but extensions must be allowed by the receiver
- Universal, i.e. only one interpreter to audit and verify
- Triviality makes it difficult to explain

Forth–Style Communication

Requirements for secure communication (secure as in “no exploitation through misinterpretation”)

- Extremely simple interpreter
- Extensible, but extensions must be allowed by the receiver
- Universal, i.e. only one interpreter to audit and verify
- Triviality makes it difficult to explain

Forth–Style Communication

Requirements for secure communication (secure as in “no exploitation through misinterpretation”)

- Extremely simple interpreter
- Extensible, but extensions must be allowed by the receiver
- Universal, i.e. only one interpreter to audit and verify
- Triviality makes it difficult to explain

Basics

- Five data types: Integer (64 bits signed+unsigned), flag, string (generic byte array), IEEE double float, objects
- Instructions and data encoding derived from Protobuf (7 bits per byte, MSB=1 means "data continues", most significant part first)
- Four stacks: integer, float, objects, strings
- endwith and endcmd for ending object message blocks and commands
- oswap to transfer the current object to the object stack, to be inserted in the outer object
- words for reflection (words are listed with token number, identifier and stack effect to make automatic bindigs possible)

Basics

- Five data types: Integer (64 bits signed+unsigned), flag, string (generic byte array), IEEE double float, objects
- Instructions and data encoding derived from Protobuf (7 bits per byte, MSB=1 means “data continues”, most significant part first)
- Four stacks: integer, float, objects, strings
- endwith and endcmd for ending object message blocks and commands
- oswap to transfer the current object to the object stack, to be inserted in the outer object
- words for reflection (words are listed with token number, identifier and stack effect to make automatic bindigs possible)

Basics

- Five data types: Integer (64 bits signed+unsigned), flag, string (generic byte array), IEEE double float, objects
- Instructions and data encoding derived from Protobuf (7 bits per byte, MSB=1 means “data continues”, most significant part first)
- Four stacks: integer, float, objects, strings
- endwith and endcmd for ending object message blocks and commands
- oswap to transfer the current object to the object stack, to be inserted in the outer object
- words for reflection (words are listed with token number, identifier and stack effect to make automatic bindigs possible)

Basics

- Five data types: Integer (64 bits signed+unsigned), flag, string (generic byte array), IEEE double float, objects
- Instructions and data encoding derived from Protobuf (7 bits per byte, MSB=1 means "data continues", most significant part first)
- Four stacks: integer, float, objects, strings
- endwith and endcmd for ending object message blocks and commands
- oswap to transfer the current object to the object stack, to be inserted in the outer object
- words for reflection (words are listed with token number, identifier and stack effect to make automatic bindigs possible)

Basics

- Five data types: Integer (64 bits signed+unsigned), flag, string (generic byte array), IEEE double float, objects
- Instructions and data encoding derived from Protobuf (7 bits per byte, MSB=1 means "data continues", most significant part first)
- Four stacks: integer, float, objects, strings
- endwith and endcmd for ending object message blocks and commands
- oswap to transfer the current object to the object stack, to be inserted in the outer object
- words for reflection (words are listed with token number, identifier and stack effect to make automatic bindigs possible)

Basics

- Five data types: Integer (64 bits signed+unsigned), flag, string (generic byte array), IEEE double float, objects
- Instructions and data encoding derived from Protobuf (7 bits per byte, MSB=1 means “data continues”, most significant part first)
- Four stacks: integer, float, objects, strings
- endwith and endcmd for ending object message blocks and commands
- oswap to transfer the current object to the object stack, to be inserted in the outer object
- words for reflection (words are listed with token number, identifier and stack effect to make automatic bindigs possible)

Why binary encoding?

- Faster and simpler to parse (simpler means smaller attack vector)
- Ability to enter commands on the fly in text form through a frontend interpreter still exists
- Debugging with a de–tokenizer is also very easy
- Object–oriented approach makes writing application–specific logic extremely simple

Why binary encoding?

- Faster and simpler to parse (simpler means smaller attack vector)
- Ability to enter commands on the fly in text form through a frontend interpreter still exists
- Debugging with a de–tokenizer is also very easy
- Object–oriented approach makes writing application–specific logic extremely simple

Why binary encoding?

- Faster and simpler to parse (simpler means smaller attack vector)
- Ability to enter commands on the fly in text form through a frontend interpreter still exists
- Debugging with a de–tokenizer is also very easy
- Object–oriented approach makes writing application–specific logic extremely simple

Why binary encoding?

- Faster and simpler to parse (simpler means smaller attack vector)
- Ability to enter commands on the fly in text form through a frontend interpreter still exists
- Debugging with a de–tokenizer is also very easy
- Object–oriented approach makes writing application–specific logic extremely simple

Why a programming language as data?

Lemma: every glue logic will become Turing complete

- Implement only the things you need — but you shouldn't have to implement more than *one* generic interpreter
- Typical idea of sending remote procedure calls: serialize the entire object (with subobjects), and call a function on that object
- Net2o idea (derived from ONF): Keep the entire object synchronized by sending only the changes to it — these changes are simple messages (setters)
- This allows multi–message passing, and reduces latency

Why a programming language as data?

Lemma: every glue logic will become Turing complete

- Implement only the things you need — but you shouldn't have to implement more than *one* generic interpreter
- Typical idea of sending remote procedure calls: serialize the entire object (with subobjects), and call a function on that object
- Net2o idea (derived from ONF): Keep the entire object synchronized by sending only the changes to it — these changes are simple messages (setters)
- This allows multi-message passing, and reduces latency

Why a programming language as data?

Lemma: every glue logic will become Turing complete

- Implement only the things you need — but you shouldn't have to implement more than *one* generic interpreter
- Typical idea of sending remote procedure calls: serialize the entire object (with subobjects), and call a function on that object
- Net2o idea (derived from ONF): Keep the entire object synchronized by sending only the changes to it — these changes are simple messages (setters)
- This allows multi–message passing, and reduces latency

Why a programming language as data?

Lemma: every glue logic will become Turing complete

- Implement only the things you need — but you shouldn't have to implement more than *one* generic interpreter
- Typical idea of sending remote procedure calls: serialize the entire object (with subobjects), and call a function on that object
- Net2o idea (derived from ONF): Keep the entire object synchronized by sending only the changes to it — these changes are simple messages (setters)
- This allows multi–message passing, and reduces latency

Security

Lemma: every sufficiently complex format can be exploited

Therefore stick to a very simple format, i.e.: simplify and factor the code

Interpreter

```
: cmd@ ( -- u )
  buf-state 2@ over + >r p@+ r> over - buf-state 2! 64>n ;
: n>cmd ( n -- addr )  cells >r
  o IF  token-table  ELSE  setup-table  THEN
  $@ r@ u<= IF  net2o-crash  THEN  r> + ;
: cmd-dispatch ( addr u -- addr' u' )  buf-state 2!
  cmd@ n>cmd @ ?dup IF  execute  ELSE  net2o-crash  THEN
  buf-state 2@ ;
: cmd-loop ( addr u -- )
  BEGIN  cmd-dispatch dup 0<= UNTIL  2drop ;
```

Security

Lemma: every sufficiently complex format can be exploited

Therefore stick to a very simple format, i.e.: simplify and factor the code

Interpreter

```
: cmd@ ( -- u )
  buf-state 2@ over + >r p@+ r> over - buf-state 2! 64>n ;
: n>cmd ( n -- addr )  cells >r
  o IF  token-table  ELSE  setup-table  THEN
  $@ r@ u<= IF  net2o-crash  THEN  r> + ;
: cmd-dispatch ( addr u -- addr' u' )  buf-state 2!
  cmd@ n>cmd @ ?dup IF  execute  ELSE  net2o-crash  THEN
  buf-state 2@ ;
: cmd-loop ( addr u -- )
  BEGIN  cmd-dispatch dup 0<= UNTIL  2drop ;
```

Reading Files

reading three files

```
0 lit, file-id "net2o.fs" $, 0 lit,
open-file <req-file get-size get-stat req> endwith
1 lit, file-id "data/2011-05-13_11-26-57-small.jpg" $, 0 lit,
open-file <req-file get-size get-stat req> endwith
2 lit, file-id "data/2011-05-20_17-01-12-small.jpg" $, 0 lit,
open-file <req-file get-size get-stat req> endwith
```

Reading Files: Reply

reading three files: replies

```
0 lit, file-id 12B9A lit, set-size
    138D607CB83D0F06 lit, 1A4 lit, set-stat endwith
1 lit, file-id 9C65C lit, set-size
    13849CAE1F3B6EA8 lit, 1A4 lit, set-stat endwith
2 lit, file-id 9D240 lit, set-size
    13849CAE2643FDCC lit, 1A4 lit, set-stat endwith
```

Messages

messages

```
msg 13977C927BF7F1AA lit, msg-at  "Hi Bob!" $, msg-text
    85" Z(&3*>qxl*bWM*DUCA-Mf9N~u;<ddcW0C<XR)ezh?=jmn7zq4RFduAe=a
    $, msg-sig endwith
85" e}&3&Kep3Im`T3?tIU=8fs>4=(C`Uic<rhs{(J`k&c5k8{H2^0*}`rV0(F3e"
$, push-$ push' nest 0 lit, ok?
```

Structured Text a la HTML

HTML–like structured text

```
body
   p "Some text with " text
       bold "bold" text oswap add
       " markup" text
   oswap add
   li
       ul "a bullet point" text oswap add
       ul "another bullet point" text oswap add
   oswap add
oswap add
```

Literature&Links

- Bernd Paysan
  net2o fossil repository
  http://fossil.net2o.de/net2o/