xh

Spencer Tipping

June 17, 2014

Contents

Part I

design

Chapter 1

constraints

xh is designed to be a powerful and ergonomic interface to multiple systems, many of which are remote. As such, it's subject to programming language, shell, and distributed-systems constraints:

1. *realprog* xh will be used for real programming. (*initial assumption*)

2. *shell* xh will be used as a shell. (*initial assumption*)

3. *distributed* xh will be used to manage any machine on which you have a login, which could be hundreds or thousands. (*initial assumption*)

4. *noroot* You will not always have root access to machines you want to use, and they may have different architectures. (*initial assumption*)

5. *ergonomic* xh should approach the limit of ergonomic efficiency as it learns more about you. (*initial assumption*)

6. *security* xh should never compromise your security, provided you understand what it's doing. (*initial assumption*)

7. *webserver* It should be possible to write a "hello world" HTTP server on one line.

    - *initial assumption*
    - *realprog* 1
    - *shell* 2

8. *liveprev* It should be possible to preview the evaluation of any well-formed expression without causing side-effects.

    - *initial assumption*
    - *shell* 2
    - *ergonomic* 5

- *nodebug* 11

9. *notslow* xh should never cause a dealbreaking performance problem.
   - *initial assumption*
   - *realprog* 1
   - *ergonomic* 5

10. *unreliable* Connections between machines may die at any time, and remain down for arbitrarily long. xh must never become unresponsive when this happens, and any data coming from those machines should block until it is available again (i.e. xh's behavior should be invariant with connection failures).
    - *initial assumption*
    - *realprog* 1
    - *shell* 2
    - *distributed* 3

11. *nodebug* Debugging should require little or no effort; all error cases should be trivially obvious.
    - *initial assumption*
    - *realprog* 1
    - *distributed* 3
    - *ergonomic* 5

12. *database* An xh instance should trivially function as a database; there should be no distinction between data in memory and data on disk.
    - *initial assumption*
    - *realprog* 1
    - *ergonomic* 5
    - *nodebug* 11
    - *no-oome* 19
    - *notslow* 9

13. *prediction* xh should use every keystroke to build/refine a model it uses to predict future keystrokes and commands. (*ergonomic* 5)

14. *history* The likelihood that xh forgets anything from your command history should be inversely proportional to the amount of effort required to retype/recreate it.
    - *ergonomic* 5

- *prediction* 13

15. *anonymous* xh must provide a way to accept input and execute commands without updating its prediction model. (*security* 6)

16. *pastebin* xh should be able to submit an encrypted version of its current state to HTTP services like Github gists or pastebin.

    - *ergonomic* 5
    - *security* 6
    - *unreliable* 10
    - *selfinstall* 52
    - *wwwinit* 53

17. *likeshell* xh-script needs to feel like a regular shell for most purposes. (*shell* 2)

18. *imperative* xh-script should be fundamentally imperative.

    - *realprog* 1
    - *shell* 2
    - *likeshell* 17

19. *no-oome* xh must never run out of memory or swap pages to disk, regardless of what you tell it to do.

    - *realprog* 1
    - *shell* 2
    - *notslow* 9
    - *ergonomic* 5

20. *nonblock* xh must respond to every keystroke within 20ms; therefore, SSH must be used only for nonblocking RPC requests (i.e. the shell always runs locally).

    - *shell* 2
    - *notslow* 9
    - *ergonomic* 5

21. *remotestuff* All resources, local and remote, must be uniformly accessible; i.e. autocomplete, filename substitution, etc, must all just work (up to random access, which is impossible without FUSE or similar).

    - *shell* 2
    - *distributed* 3
    - *ergonomic* 5

22. *prefix* xh-script uses prefix notation. (*shell* 2)

23. *quasiquote* xh-script quasiquotes values by default. (*shell* 2)

24. *unquote* xh-script defines an unquote operator.

    - *shell* 2
    - *quasiquote* 23

25. *datastruct* The xh runtime provides real, garbage-collected data structures. (*realprog* 1)

26. *quotestruct* Every xh data structure has a quoted form.

    - *datastruct* 25
    - *shell* 2
    - *nodebug* 11
    - *liveprev* 8

27. *printstruct* Every xh data structure can be losslessly serialized.

    - *shell* 2
    - *distributed* 3
    - *database* 12
    - *quotestruct* 26
    - *varsinrc* 55
    - *imagemerging* 58

28. *immutable* Data structures have no identity and therefore are immutable.

    - *distributed* 3
    - *printstruct* 27

29. *opaques* xh-script must have access to machine-specific opaque resources like PIDs and file handles.

    - *realprog* 1
    - *shell* 2

30. *mutablesyms* Each xh instance should implement a mutable symbol table with weak reference support, subject to semi-conservative distributed garbage collection.

    - *immutable* 28
    - *opaques* 29
    - *no-oome* 19

- *heap* 46

31. *stateown* Every piece of mutable state, including symbol tables, must have at most one authoritative copy (mutable state within xh is managed by a CP system).
    - *unreliable* 10
    - *opaques* 29
    - *mutablesyms* 30
    - *threadmobility* 49
32. *checkpoint* An xh instance should be able to save checkpoints of itself in case of failure. If you do this, xh becomes an AP system.
    - *unreliable* 10
    - *stateown* 31
33. *lazy* xh's evaluator must support some kind of laziness.
    - *realprog* 1
    - *no-oome* 19
    - *remotestuff* 21
    - *notslow* 9
34. *printlazy* Lazy values must have well-defined quoted forms and be losslessly serializable.
    - *quotestruct* 26
    - *printstruct* 27
    - *lazy* 33
    - *threadmobility* 49
    - *heap* 46
35. *introspectlazy* All lazy values must be subject to introspection to identify why they haven't been realized.
    - *nodebug* 11
    - *notslow* 9
    - *unreliable* 10
    - *nonblock* 20
    - *lazy* 33
    - *threadscheduler* 48
36. *abstract* xh must be able to partially evaluate expressions that contain unknown quantities.

- *liveprev* 8
- *lazy* 33
- *introspectlazy* 35
- *printlazy* 34

37. *code=data* xh-script code should be a reasonable data storage format.
    - *shell* 2
    - *abstract* 36
38. *selfparse* xh-script must contain a library to parse itself.
    - *code=data* 37
39. *homoiconic* xh-script must be homoiconic.
    - *code=data* 37
    - *selfparse* 38
    - *selfhost* 43
    - *abstractstruct* 45
40. *xh2c* xh should be able to compile any function to C, compile it if the host has a C compiler, and transparently migrate execution into this process.
    - *realprog* 1
    - *threadmobility* 49
    - *notslow* 9
41. *xh2perl* xh should be able to compile any function to Perl rather than interpreting its execution.
    - *realprog* 1
    - *noroot* 4
    - *notslow* 9
42. *xh2js* xh should be able to compile any function to Javascript so that browser sessions can transparently become computing nodes.
    - *realprog* 1
    - *distributed* 3
    - *notslow* 9
43. *selfhost* xh should follow a bootstrapped self-hosting runtime model.
    - *xh2c* 40
    - *xh2perl* 41

- *xh2js* 42
- *abstractstruct* 45

44. *dynamiccompiler* xh-script should be executed by a profiling/tracing dynamic compiler that automatically compiles certain pieces of code to alternative forms like Perl or C. (*notslow* 9)

45. *abstractstruct* The xh compiler should optimize data structure representations for the backend being targeted.

    - *notslow* 9
    - *threadmobility* 49
    - *dynamiccompiler* 44

46. *heap* xh needs to implement its own heap and memory manager, and swap values to disk without blocking.

    - *realprog* 1
    - *no-oome* 19
    - *database* 12
    - *inperl* 56

47. *threading* xh should implement its own threading model to accommodate blocked IO requests.

    - *shell* 2
    - *distributed* 3
    - *webserver* 7
    - *lazy* 33
    - *heap* 46

48. *threadscheduler* xh threads should be subject to scheduling that reflects the user's priorities.

    - *shell* 2
    - *distributed* 3
    - *lazy* 33
    - *threading* 47

49. *threadmobility* Running threads must be transparently portable between machines and compiled backends.

    - *distributed* 3
    - *threading* 47
    - *dynamiccompiler* 44

- *abstractstruct* 45
- *threadscheduler* 48

50. *refaffinity* All machine-specific references must encode the machine for which they are defined.

    - *opaques* 29
    - *threadmobility* 49

51. *uniqueid* Every xh instance must have a unique ID, ideally one that can be typed easily.

    - *ergonomic* 5
    - *refaffinity* 50

52. *selfinstall* xh needs to be able to self-install on remote machines with no intervention (assuming you have a passwordless SSH connection).

    - *distributed* 3
    - *noroot* 4

53. *wwwinit* You should be able to upload your xh image to a website and then install it with a command like this: `curl me.com/xh | perl`.

    - *distributed* 3
    - *noroot* 4

54. *selfmodifying* Your settings should be present as soon as you download your image, so the image must be self-modifying and contain your settings.

    - *distributed* 3
    - *ergonomic* 5
    - *prediction* 13
    - *selfinstall* 52
    - *wwwinit* 53

55. *varsinrc* Your settings should be able to contain any value you can create from the REPL (with the caveat that some are defined only with respect to a specific machine).

    - *realprog* 1
    - *shell* 2
    - *ergonomic* 5
    - *datastruct* 25
    - *wwwinit* 53

56. *inperl* xh should probably be written in Perl 5.

    - *distributed* 3
    - *noroot* 4
    - *selfinstall* 52
    - *wwwinit* 53
    - *selfmodifying* 54

57. *perlcoreonly* xh can't have any dependencies on CPAN modules, or anything else that isn't in the core library.

    - *distributed* 3
    - *noroot* 4
    - *selfinstall* 52

58. *imagemerging* It should be possible to address variables defined within xh images (as files or network locations).

    - *selfmodifying* 54
    - *varsinrc* 55

59. *sshrpc* xh's RPC protocol must work via stdin/out communication over an SSH channel to a remote instance of itself.

    - *distributed* 3
    - *security* 6
    - *selfinstall* 52
    - *nonblock* 20
    - *remotestuff* 21

60. *rpcmulti* xh's RPC protocol must support request multiplexing.

    - *distributed* 3
    - *notslow* 9
    - *nonblock* 20
    - *remotestuff* 21
    - *lazy* 33
    - *sshrpc* 59

61. *hostswitch* Two xh servers on the same host should automatically connect to each other. This allows a server-only machine to act as a VPN.

    - *distributed* 3
    - *noroot* 4

- *sshrpc* 59
- *transitive* 63

62. *domainsockets* xh should create a UNIX domain socket to listen for other same-machine instances.

    - *security* 6
    - *hostswitch* 61

63. *transitive* xh's network topology should forward requests transitively.

    - *distributed* 3
    - *noroot* 4
    - *sshrpc* 59

64. *routing* xh should implement a network optimizer that responds to observations it makes about latency and throughput.

    - *notslow* 9
    - *sshrpc* 59
    - *transitive* 63

Chapter 2

xh-script

These constraints are based on the ones in chapter 1.

1. *xhs.eval-identities* Evaluation of any expression may happen at any time; the only scheduling constraint is the realization of lazy expressions, whose status is visible by looking at their quoted forms. Therefore, the evaluator is, to some degree, associative, commutative, and idempotent.

    - *initial assumption*
    - *distributed* 3 above
    - *nodebug* 11 above
    - *liveprev* 8 above
    - *nonblock* 20 above
    - *lazy* 33 above
    - *introspectlazy* 35 above
    - *abstract* 36 above

2. *xhs.relational* Relational evaluation is possible by using `amb`, which returns any of the given presumably-equivalent values. xh-script is relational and invertible, though inversion is not always lossless and may produce perpetually-unresolved unknowns representing degrees of freedom.

    - *initial assumption*
    - *nodebug* 11 above
    - *lazy* 33 above
    - *introspectlazy* 35 above
    - *abstract* 36 above
    - *selfhost* 43 above
    - *abstractstruct* 45 above

- *threadscheduler* 48 above
- *xhs.eval-identities* 1

3. *xhs.bestfirst* Due to functions like `amb`, evaluation proceeds as a best-first search through the space of values. You can influence this search by defining the `abstraction` relation for a particular class of expressions.

   - *notslow* 9 above
   - *xhs.relational* 2

4. *xhs.nocut* Unlike Prolog, xh defines no cut primitive. You should use `abstraction` to locally grade the search space instead.

   - *nodebug* 11 above
   - *xhs.eval-identities* 1
   - *xhs.bestfirst* 3

5. *xhs.unquote-structure* The unquoting operators `$`, `$@`, `()`, and `@()` each preserve some aspect of structure:

   - `$` and `()` preserve list structure and atom count, and strictly decrease quotation level.
   - `$@` and `@()` preserve list levels while varying atom count, and strictly decrease quotation level.
   - `$!` and `!()` locally preserve list structure, atom count, and quotation level. These are the only forms that won't block when expanding unrealized lazy values.

   Operators that decrease quotation level will block if the result requires quoting to represent fully.

   - *initial assumption*
   - *realprog* 1 above
   - *unquote* 24 above
   - *notslow* 9 above)

6. *xhs.stackscope* All scoping is done by passing a second argument to `unquote`; this enables variable resolution during the unquoting operation.

   - *initial assumption*
   - *unquote* 24 above
   - *mutablesyms* 30 above
   - *xhs.eval-identities* 1

7. *xhs.noshadow* Variable shadowing is impossible.

- *xhs.eval-identities* 1
- *xhs.stackscope* 6

8. *xhs.unquote-parse* Unquoting and structural parsing are orthogonal operations provided by `unquote` and `read`, respectively.

   - *quotestruct* 26 above
   - *introspectlazy* 35 above
   - *xhs.eval-identities* 1
   - *xhs.unquote-structure* 5

9. *xhs.runtimereceiver* Whether via RPC or locally, statements issued to an xh runtime can be interpreted as messages being sent to a receiver; the reply is sent along whatever continuation is specified. The runtime doesn't differentiate between local and remote requests, including those made by functions.

   - *imperative* 18 above
   - *threading* 47 above
   - *threadmobility* 49 above
   - *stateown* 31 above

10. *xhs.namespaces* Functions and variables exist in separate namespaces.

    - *likeshell* 17 above
    - *unquote* 24 above
    - *xhs.stackscope* 6
    - *xhs.runtimereceiver* 9

11. *xhs.fnliterals* Function literals are self-invoking when used as messages. (*xhs.namespaces* 10)

12. *nocallcc* Continuations are simulated in terms of lazy evaluation, but are never first-class.

    - *dynamiccompiler* 44 above
    - *introspectlazy* 35 above
    - *abstract* 36 above
    - *xhs.runtimereceiver* 9

13. *xhs.transientdefs* Some definitions are "transient," in which case they are used to resolve blocked lazy values but then may be discarded at any point.

    - *distributed* 3 above

- *no-oome* 19 above
- *lazy* 33 above
- *xhs.runtimereceiver* 9

14. *xhs.globaldefs* Global definitions can apply to values at any time, and to values on different machines (i.e. their existence is broadcast).

    - *lazy* 33 above
    - *xhs.transientdefs* 13

15. *xhs.nomacros* Syntactic macros cannot exist because invocation commutes with expansion, but functions may operate on terms whose values are undefined.

    - *xhs.eval-identities* 1
    - *xhs.unquote-structure* 5

16. *xhs.noerrors* Errors cannot exist, but are represented by lazy values that contain undefined quantities that will never be realized. These undefined quantities are the unevaluated backtraces to the error-causing subexpressions.

    - *nodebug* 11 above
    - *lazy* 33 above
    - *abstract* 36 above
    - *xhs.eval-identities* 1

17. *xhs.destructuring* Any value can be used as a destructuring bind pattern.

    - *initial assumption*
    - *xhs.relational* 2

18. *xhs.ambdestructure* `(amb)` can be used to destructure values, and it behaves as a disjunction.

    - *initial assumption*
    - *xhs.relational* 2
    - *xhs.destructuring* 17

19. *xhs.dof* Degrees of freedom within an inversion are represented by abstract values that will prevent the result from being realized. They are visible as unknowns within the quoted form.

    - *initial assumption*
    - *xhs.relational* 2

20. *xhs.se-axioms* Side effects and axioms are the same thing in xh. Once it commits to a side-effect, it must always assume that it happened (since it did). In particular, this means that imperative forms like `(def)` are actually ways to assume new ground truths.

    - *initial assumption*
    - *imperative* 18
    - *xhs.relational* 2
    - *xhs.globaldefs* 14

21. *xhs.virtualization* Every side effect can be replaced by a temporary assumption that models the effect. If you do this, you're replacing an axiom with a hypothesis.

    - *initial assumption*
    - *xhs.se-axioms* 20

Chapter 3

runtime

xh-script operates within a hosting environment that manages things like memory allocation and thread/evaluation scheduling. Beyond this, we also need a quoted-value format that's more efficient than doing a bunch of string manipulation (*xhr.representation* 4, *xhr.flatcontainers* 5, *xhr.deduplication* 6).

1. *xhr.priorityqueue* Evaluation always happens as a process of pulling expressions from a priority queue.

   - *initial assumption*
   - *xhs.relational* 2
   - *xhs.bestfirst* 3

2. *xhr.prioritytracing* Every expression in the queue knows its "origin" for scheduling purposes.

   - *xhs.bestfirst* 3
   - *xhr.priorityqueue* 1

3. *xhr.staticinline* Function compositions should be added as derived definitions to minimize the number of symbol-table lookups per unit rewriting distance.

   - *initial assumption*
   - *notslow* 9
   - *xhs.relational* 2

4. *xhr.representation* Every quasiquoted form with variant pieces should be represented as a separate instantiable class.

   - *initial assumption*
   - *quasiquote* 23
   - *notslow* 9

- *xhs.eval-identities* 1

5. *xhr.flatcontainers* Quasiquoted structures are profiled for the distributions of their children (upon expansion); for strongly nonuniform distributions, specialized flattened container types are generated.

   - *initial assumption*
   - *quasiquote* 23
   - *notslow* 9
   - *xhs.eval-identities* 1
   - *xhr.staticinline* 3
   - *xhr.representation* 4

6. *xhr.deduplication* Every independent value within a quasiquoted form should be referred to by a structural signature, in our case SHA-256. This trivially causes strings, and by extension execution paths, to be deduplicated. Because we assume no hash collisions, xh string values have no defined instance affinity (apropos of *refaffinity* 50).

   - *heap* 46
   - *xhr.staticinline* 3
   - *xhr.representation* 4
   - *xhr.hinting* 8

7. *xhr.pointerentropy* 256 bits is sufficient to encode any pointer.

   - *initial assumption*
   - *uniqueid* 51
   - *refaffinity* 50
   - *xhr.deduplication* 6

8. *xhr.hinting* Expressions should be hinted with tags that track and influence their paths through the search space. The optimizer uses machine learning against these tags to predict successful search strategies.

   - *initial assumption*
   - *notslow* 9
   - *xhs.bestfirst* 3
   - *xhs.nocut* 4

9. *xhr.hashing* The runtime should use some type of masked hashing strategy (or other decision tree) to minimize the expected resolution time for each expression.

   - *initial assumption*

- *xhr.hinting* 8

10. *xhr.transientprediction* Many functions will end up returning lazy values, and most of the time those lazy values will eventually be realized. The runtime should have some expectation of which lazy sub-values will be realized, and with what probability; this influences its search strategy in the future.

    - *initial assumption*
    - *notslow* 9
    - *xhs.transientdefs* 13
    - *xhs.bestfirst* 3
    - *xhr.hinting* 8

11. *xhr.override* The user must be able to completely override any strategy preferences the runtime has. The runtime can be arbitrarily wrong and the user can be arbitrarily right.

    - *initial assumption*
    - *xhs.bestfirst* 3
    - *xhr.hinting* 8
    - *xhr.transientprediction* 10

12. *xhr.externalstrategy* The xh runtime does not itself define the evaluation strategy, nor does it internally observe things; this is done as part of the evaluation functions in the standard library. The only thing the xh runtime provides is a scheduled/prioritized event loop.

    - *initial assumption*
    - *abstract* 36
    - *code=data* 37
    - *xhr.override* 11

Part II

base implementation

Chapter 4

self-replication

Listing 4.1 boot/xh-header

```perl
#!/usr/bin/env perl
BEGIN {eval(our $xh_bootstrap = q{
# xh | https://github.com/spencertipping/xh
# Copyright (C) 2014, Spencer Tipping
# Licensed under the terms of the MIT source code license

# For the benefit of HTML viewers (long story):
# <body style='display:none'>
# <script src='http://spencertipping.com/xh/page.js'></script>
use 5.014;
package xh;
our %modules;
our @module_ordering;
our %eval_numbers = (1 => '$xh_bootstrap');

sub with_eval_rewriting(&) {
  my @result = eval {$_[0]->(@_[1..$#_])};
  die $@ =~ s/\(eval (\d+)\)/$eval_numbers{$1}/egr if $@;
  @result;
}

sub named_eval {
  my ($name, $code) = @_;
  $eval_numbers{$1 + 1} = $name if eval('__FILE__') =~ /\(eval (\d+)\)/;
  with_eval_rewriting {eval $code; die $@ if $@};
}

our %compilers = (pl => sub {
  my $package = $_[0] =~ s/\./::/gr;
```

```
  eval {named_eval $_[0], "{package ::$package;\n$_[1]\n}"};
  die "error compiling module $_[0]: $@" if $@;
});

sub defmodule {
  my ($name, $code, @args) = @_;
  chomp($modules{$name} = $code);
  push @module_ordering, $name;
  my ($base, $extension) = split /\.(\w+$)/, $name;
  die "undefined module extension ’$extension’ for $name"
    unless exists $compilers{$extension};
  $compilers{$extension}->($base, $code, @args);
}

chomp($modules{bootstrap} = $::xh_bootstrap);
undef $::xh_bootstrap;
```

At this point we need a way to reproduce the image. Since the bootstrap code is already stored, we can just wrap it and each defined module into an appropriate BEGIN block.

Listing 4.2 boot/xh-header (continued)

```
sub image {
  join "\n", "#!/usr/bin/env perl",
             "BEGIN {eval(our \$xh_bootstrap = <<’_’)}",
             $modules{bootstrap},
             ’_’,
             map("BEGIN {xh::defmodule(’$_’, <<’_’)}\n$modules{$_}\n_",
                 @module_ordering),
             "xh::main::main;\n__DATA__";
}
})}
```

Chapter 5

reader

xh-script has a reader just like Lisp does. This makes it easier to factor the runtime: the reader is invariant with the semantics of the language under any given interpreter/compiler. For simplicity, this reader does not stream its output. Instead, it emits a full quoted data structure hierarchy in OO format.

Listing 5.1 src/v.pl

```
BEGIN {xh::defmodule('xh::v.pl', <<'_')}
-- include src/v/type-definition.pl
-- include src/v/reader.pl
_
```

Listing 5.2 src/v/type-definition.pl

```
use parent qw/Exporter/;
our @EXPORT_OK = qw/parse/;

sub new {
  my ($class, $type, $tag, @values) = @_;
  bless [$type, $tag, @values], $class;
}
```

Listing 5.3 src/v/reader.pl

```
sub parse {
  my @return_values;
  my @context = (\@return_values);

  while ($_[0] =~ /\G (?: \s* | \#.*)*
                      (?: (?<tag> (?: [^\s()\[\]{}"'\\] | \\.)+)?
                          (?: (?<listopen>   \()
                            | (?<vectoropen> \[)
                            | (?<mapopen>    \{)
```

```
                            | "(?<dstring>   (?:[^"\\]*|\\[\s\S]))"
                            | '(?<sstring>   (?:[^'\\]*|\\[\s\S]))')
                        | (?<word>        (?: [^\s()\[\]{}'"\\] | \\.)+)
                        | (?<listclose>   \))
                        | (?<vectorclose> \])
                        | (?<mapclose>    \}))/xmg) {
    my $opener = $+{listopen} // $+{vectoropen} // $+{mapopen};
    if (defined $+{word}) {
      push @{$context[-1]}, $+{word};
    } elsif (defined $+{dstring}) {
      push @{$context[-1]}, xh::v->new('"', $+{tag}, $+{dstring});
    } elsif (defined $+{sstring}) {
      push @{$context[-1]}, xh::v->new("'", $+{tag}, $+{sstring});
    } elsif (defined $opener) {
      my $new_container = xh::v->new($opener, $+{tag});
      push @{$context[-1]}, $new_container;
      push @context, $new_container;
    } elsif (defined($+{listclose} // $+{vectorclose} // $+{mapclose})) {
      my $popped = pop @context;
      push @{$context[-1]}, $popped;
    }
  }
  @return_values;
}
```