XBONE APPLICATION PROGRAMMERS INTERFACE

Xbone Overlay Language (XOL) Specification

Gregory G. Finn, Joseph D. Touch

Nov. 2001
ISI-TR-2001-549

Abstract

The Xbone system 2.0 release provides an API. This API is implemented by the Overlay Manager (OM) and is the method by which the OM receives user commands and returns responses. This corrects a difficulty encountered with the previous command interface to the OM, which was both brittle, difficult to extend and could only be accessed via a web GUI.

To allow for future expansion and because the description of an overlay or system of overlays can be complex, a grammar for describing Xbone commands and overlays has been developed. The OM contains a compiler that implements this grammar. Commands sent to the API must obey this grammar. Responses made by the OM via the API also obey this grammar. This report describes the Xbone API grammar and the corresponding implementation interface details.

Overview

The XBone Overlay Manager (OM) listens to a well-known TCP command port. The XBone API consists of grammar rules governing message exchange across that port. When a connection to that API port is opened, the OM expects to receive a program that specifies an action or set of actions. The OM replies in kind over that same connection port. That port is then closed, whereupon the OM listens for another connection.

A grammar defines the programming language used to communicate across the XBone API. The grammar provides access to a set of commands implemented at one or the other end of the API communication path. It is generally convenient to think of the OM as the server in an API client/server message exchange, although the OM can in some cases be a client.

The OM treats any message received over its API port as a program and attempts to compile it. If compilation is successful, the OM attempts to 'execute' that program. The OM may respond likewise to the caller with program text that either contains a compiler error or the result of program execution. XBone API programs must also abide by the Xbone's current implementation restrictions.

The API grammar rules are relatively simple and describe a set of legal commands and their responses. However, one API command, "create_overlay", is complex. This command accepts as its argument a program that describes an overlay. The Xbone Overlay Language (XOL) is used to describe an overlay or system of overlays.

The compiler that parses both API and XOL grammars is created using the Perl RecDescent module, which is available at the CPAN library site. Both RecDescent, the grammar and semantic action routines are included in the XBone distribution. These files are not needed unless one modifies the compiler.

While the grammar defines the program syntax, there are programs that are syntactically legal but semantically incorrect. At parse time, many of these inconsistences are detected and result in failed parse. In addition, the Xbone as released only implements a subset of the features of the langauge [see the Appendix].

Grammar Conventions

RecDescent allows Perl regular expressions to be used in right-hand sides. This simplifies the definition of strings. RecDescent also allows the following suffixes to appear in the rules:

```
      (?) ----- Match one or zero times
      (s) ----- Match one or more times
      (s?) ---- Match zero or more times
      (N) ----- Match exactly N times, for N a positive integer
      (N..M) -- Match between N and M times
      (..N) --- Match between 1 and N times
      (N..) --- Match at least N times
```

Thus, if OVERLAY(1..) appears in the right-hand side of a rule, one or more OVERLAY expressions may appear in sequence at that point. Several of those suffix forms appear in the API language specification.

For more details on RecDescent itself, consult the Perl CPAN site.

Language variables here are entirely in upper case for ease of reading. All keywords are entirely --> lower case <--. The keyword IPV4 is not acceptable where ipv4 is called for in the grammar.

Although keywords are case sensitive, there are no case restrictions on user-supplied strings. However, syntactic resrictions on domain system (DNS) names, numeric addresses and similarly defined fields are enforced.

Compiler Conventions

The API/XOL compiler returns undef if a syntactic error occurs. Otherwise, it returns one of two values.

If the parse is successful a value is returned. This value will be either a pointer to the hash data structure, which contains the compilation result, or a semantic error string.

Thus, a completely successful parse results in a defined return value that is a reference to a hash. The Perl language function ref() is used to make this determination.

For those wishing to build a new front-end to the Xbone system rather than using the GUI supplied by the distribution, the code fragment below indicates how to use the API parser. The initial grammar rule for the API language is named "API". The compiler invocation $APIparser->API(program-string) begins parsing at the API rule.

```
use XB_API_parser;
my $APIparser = XB_API_parser->new();

$parsed_cmd = $APIparser->API($commands);

unless (!defined ($parsed_cmd) or ref (\$parsed_cmd) eq 'SCALAR')
  {
   # Compilation succeeded
  }
```

When creating a front-end, messages returned by the OM will be in the form of API command response programs. Compiling these responses is an easy way to ensure legality and extract needed information.

Keywords

The following are used as keywords by the API/XOL language. Use of these keywords except where indicated by the language could result in a failed parse. In particular, beware of creating CRITERION name/value pairs that use a keyword as a name.

```
3des
address_server
addresstype
any
authentication
create_overlay
create_overlay_reply
creator
des
destroyall_overlays
destroyall_overlays_reply
destroy_overlay
destroy_overlay_reply
discover_daemons
discover_daemons_reply
email
encryption
error-reply
hosts
interfaces
ipv4
ipv6
links
list_overlays
list_overlays_reply
manager
md5
monitor
name
name_server
node
none
overlay
overlay_status
overlay_status_reply
port
program
routers
security
services
sha1
system
tunnel
xbone
XboneEOC
xol
```

Strings

In the XBone API/XOL language the syntax of an argument string, (see ARGSTRING rule), allows use of unquoted, single or double-quoted strings. This is useful for specifying personal names that include spaces, dashes or apostrophes. For example: "G.G O'Hara-Pierce". The only characters not allowed are parenthesis ().

Object names in the language, (see OBJECT_NAME rule), must be alphanumeric strings, but also allow underbar "_", dash "-" and period "." as additional characters.

Where the semantic nature of the string is more restrictive, as in Email or DNS names, those restrictions are applied. If failure results, an error message is generated during compilation and will be reported in the response message.

API Grammar

What follows is the specification of the API grammar. For an example of a program that contains a 'create_overlay' command, see the Appendices.

```
  API:  '('  'xbone'  ARGSTRING  ARGSTRING  COMMAND  ')'  'XboneEOC'  END_OF_FILE

 COMMAND:  '('  'create_overlay'             CRITERION(s)  CO_PROGRAM  ')'
        |  '('  'create_overlay_reply'       CRITERION(s)  NODE(s) ')'
        |  '('  'list_overlays'              CRITERION(s)  ')'
        |  '('  'list_overlays_reply'        CRITERION  ARGSTRING(s?)  ')'
        |  '('  'overlay_status'             CRITERION(s)  ')'
        |  '('  'overlay_status_reply'       CRITERION(s)  NODE(s?)  ')'
        |  '('  'discover_daemons'           CRITERION(s)  ')'
        |  '('  'discover_daemons_reply'     CRITERION(s)  NODE(s?)  ')'
        |  '('  'destroy_overlay'            CRITERION(s)  ')'
        |  '('  'destroy_overlay_reply'      CRITERION(s)  ')'
        |  '('  'destroyall_overlays'        CRITERION(s)  ')'
        |  '('  'destroyall_overlays_reply'  CRITERION(s)  ')'
        |  '('  'host_choices'               CRITERION(s)  NODE(s)  ')'
        |  '('  'host_choices_reply'         CRITERION(s)  NODE(s)  ')'
        |  '('  'error_reply'        CRITERION(s)  ')'

  NODE:  '('  'node'  CRITERION(s)  TUNNEL(s?)  ')'

  TUNNEL:  '('  'tunnel'  CRITERION(s) ')'

  CO_PROGRAM:  '('  'program'  XOL_PROGRAM  ')'

  CRITERION:  '('  ARGSTRING ARGSTRING  ')'

  ARGSTRING:  /"[^"()]+"/        # Cannot contain embedded double quotes
           |  /'[^'()]+'/        # Cannot contain embedded single quotes
           |  /[^()\s]+/

  END_OF_FILE:  /\Z/
```

Semantic Details

Each command sent to the OM will contain a set of key/value pairs. There are many such pairs and those used vary according to the COMMAND variant employed. There are also repetitive elements in some reply messages.

Key/value pairs are specified using the CRITERION grammar rule. The rule groups in parenthesis a key and value in left-to-right order. Thus, "(name Fred)" assigns key=name and value=Fred.

Wherever a CRITERION(s) rule appears, one or more key/value pairs may be specified. There is no order relationship associated with the CRITERION(s) rule. However, the key fields should be unique. If they are not, only the last one is used.

A detailed description of each API, COMMAND, NODE and TUNNEL rule follows.

API Rule

```
API:  (  xbone  ARGSTRING  ARGSTRING  COMMAND  )  XboneEOC
```

Every API command is sent as a single program in the API grammar. These programs are framed at the transport layer by the start of transmission and the terminating "XboneEOC" keyword.

The OM listens on its assigned API port for a TCP connection. It reads until it sees "XboneEOC". The data read is then compiled and a response message is returned back over the same connection, which the OM then closes.

Every API command is associated with an XBone version. When a command is parsed, the result returned to the caller is normally a pointer to a proplist. That proplist contains the following key/value pairs:

```
      protocol ---  The first ARGSTRING is the Xbone protocol release
                     number.  Format is "integer.alphanumeric".
                     The proper release number is defined in the file
                     xbone/XB_Defs.pm as XB_Defs::XBONE_PROTOCOL.

       release ---  The second ARGSTRING in the API rule is the current
                     Xbone system release number.  The release number has
                     format "integer.alphanumeric", (integer is unsigned).
                     The proper release number is defined in the file
                     xbone/XB_Defs.pm as XB_Defs::XBONE_RELEASE.

       command ---  Value is a pointer to a proplist (hash) containing
                   the COMMAND data in the API rule.
```

COMMAND Rule

```
COMMAND:  (  create_overlay  CRITERION(s)  CO_PROGRAM  )

The key/value pairs of the create_overlay command are:

      auth_type -------- value is a supported authentication method,
                           currently only x509 is supported
      authentication --- value is the IPsec authentication type used for
                           this overlay, currently either md5, sha1 or none
      command ---------- Specific command, here 'create_overlay'
      creator_name ----- value is the name of the overlay creator
      creator_email ---- value is the email address of the overlay creator
      dns -------------- value is yes if DNS is being used to name nodes
                           in the overlay and no otherwise
      dynamic_routing -- value is 'yes' if dynamic routing is used and
                           'no' otherwise
      encryption ------- value is the IPsec encryption type used for this
                           overlay, currently either des, 3des or none
      hosts ------------ unsigned integer number of hosts in the overlay
      host_os ---------- value is name of OS used for hosts
      overlay_name ----- value is the name of the overlay network
      two_phase -------- value is 'yes' if two-phase invitation/commit is
                           to be used and 'no' otherwise
      routers ---------- unsigned integer number of routers in the overlay
      router_os -------- value is name of OS used for routers
      search_radius ---- value is unsigned integer number of hops from
                         OM to search outwards for discovery
      topology --------- if present, the topology to be used for this
                           overlay.  Currently, this must be either 'star',
                           'linear' or 'ring'
      user_id ---------- typically the email address of the current user
```

```
COMMAND:  (  create_overlay_reply  CRITERION(s)  NODE(s) )

The key/value pairs of the create_overlay_reply command are:

        auth_type ------ value is a supported authentication method,
                           currently only x509 is supported
        command -------- Specific command, here 'create_overlay_reply'
        creator_name --- value is the name of the overlay creator
        creator_email -- value is the email address of the overlay creator
        error ---------- value is present only when reporting an error
                           if present may be only name/value pair other
                           than command
        nodes ---------- value is a pointer to a list of NODE proplists
        overlay_ident -- value is an identifier uniquely assigned by OM
        overlay_name --- value is the name of the overlay network
        topology ------- if present it is topology type, 'star', 'ring',
                           or 'linear'.  Otherwise, topology is untyped.
        user_id -------- typically the email address of the current user


COMMAND:  (  list_overlays  CRITERION(s)  )

The trailing key/value pairs of the list_overlays command are:

        auth_type ------ value is a supported authentication method,
                           currently only x509 is supported
        command -------- Specific command, here 'list_overlays'
        user_id -------- typically the email address of the current user


COMMAND:  (  list_overlays_reply  CRITERION(?)  ARGSTRING(s?)  )

The trailing key/value pair of the list_overlays_reply command is:

        command -------- Specific command, here 'list_overlays_reply'
        error ---------- value is present only when reporting an error
                           if present may be only name/value pair other
                           than command
        overlays ------- value is a pointer to a list of ARGSTRINGs
                           that are the names of overlay networks


COMMAND:  (  overlay_status  CRITERION(s)  )

The trailing key/value pairs of the overlay_status command are:

        auth_type ------ value is a supported authentication method,
                           currently only x509 is supported
        command -------- Specific command, here 'overlay_status'
        search_radius -- value is unsigned integer number of hops from
                         OM to search outwards for discovery
        user_id -------- typically the email address of the current user
```

```
COMMAND:  (  overlay_status_reply  CRITERION(s)  NODE(s?)  )

The key/value pairs of the overlay_status_reply command are:

        auth_type ------ value is a supported authentication method,
                           currently only x509 is supported
        command -------- Specific command, here 'overlay_status_reply'
        creator_name --- value is the name of the overlay creator
        creator_email -- value is the email address of the overlay creator
   ipsec_authentication -- value is the IPsec authentication type used for
                           this overlay, currently either md5, sha1 or none
        ipsec_encryption -- value is the IPsec encryption type used for this
                           overlay, currently either des, 3des or none
        error ---------- value is present only when reporting an error
                           if present may be only name/value pair other
                           than command
        nodes ---------- value is a pointer to a list of NODE proplists
        overlay_name --- value is the name of the overlay network
        user_id -------- typically the email address of the current user


COMMAND:  (  discover_daemons  CRITERION(s)  )

The trailing key/value pairs of the discover_daemons command are:

        auth_type ------ value is a supported authentication method,
                           currently only x509 is supported
        command -------- Specific command, here 'discover_daemons'
        creator_name --- value is the name of the overlay creator
        creator_email -- value is the email address of the overlay creator
        search_radius -- value is unsigned integer number of hops from
                         OM to search outwards for discovery
        timeout -------- value is unsigned integer number of seconds to
                           wait for replies from RDs
        user_id -------- typically the email address of the current user


COMMAND:  (  discover_daemons_reply  CRITERION(s)  NODE(s?)  )

The key/value pairs of the discover_daemons_reply command are:

        auth_type ------ value is a supported authentication method,
                           currently only x509 is supported
        command -------- Specific command, here 'discover_daemons_reply'
        creator_name --- value is the name of the overlay creator
        creator_email -- value is the email address of the overlay creator
        error ---------- value is present only when reporting an error
                           if present may be only name/value pair other
                           than command
        nodes ---------- value is a pointer to a list of NODE proplists
        user_id -------- typically the email address of the current user
```

```
COMMAND:  (  destroy_overlay  CRITERION(s)  )

The trailing key/value pairs of the destroy_overlay command are:

        auth_type ----- value is a supported authentication method,
                          currently only x509 is supported
        command ------- Specific command, here 'destroy_overlay'
        overlay_name -- value is the name of the overlay network
        user_id ------- typically the email address of the current user


COMMAND:  (  destroy_overlay_reply  CRITERION(s)  )

The trailing key/value pair of the destroy_overlay_reply command is:

        command ------- Specific command, here 'destroy_overlay_reply'
        overlay_name -- value is the name of the overlay network
        error --------- value is present only when reporting an error
                          if present may be only name/value pair other
                          than command


COMMAND:  (  destroyall_overlays  CRITERION(s)  )

The trailing key/value pairs of the destroyall_overlays command are:

        auth_type -- value is a supported authentication method,
                       currently only x509 is supported
        command -------- Specific command, here 'destroyall_overlays'
        user_id ---- typically the email address of the current user


COMMAND:  (  destroyall_overlays_reply  CRITERION(s)  )

The trailing key/value pair of the destroyall_overlays_reply command is:

        command -------- Specific command, here 'destroyall_overlays_reply'
        error ---------- value is present only when reporting an error
                           if present may be only name/value pair other
                           than command


COMMAND:  (  host_choice  CRITERION(s)  NODE(s) )

The key/value pairs of the host_choice command are:

        command -------- Specific command, here 'host_choice'
```

```
COMMAND:  (  host_choice_reply  CRITERION(s)  NODE(s) )

The trailing key/value pair of the destroyall_overlays_reply command is:

        command -------- Specific command, here 'host_choice_reply'
        error ---------- value is present only when reporting an error
                           if present may be only name/value pair other
                           than command


COMMAND:  (  error_reply  CRITERION(s) )

The key/value pairs of the error_reply command are:

        command -------- Specific command, here 'error_reply'
          error ---------- Semantic or syntactic error string used to
                           indicate why an API message did not parse.
                           Generally, if an OM cannot parse an API
                     message, it returns an error_reply back to
                     API message sender over the same connection.
```

CO_PROGRAM Rule

```
CO_PROGRAM:  (  program  XOL_PROGRAM  )
```

This trailing key/value pair of the create_overlay command defines the XOL program used to describe the overlay that is to be created.

```
        program -------- Value is a reference to the XOL_PROGRAM hash.
```

NODE Rule

```
NODE:  (  node  CRITERION(s)  TUNNEL(s?)  )
```

Each NODE describes the state of a particular node (host) in an overlay. This state is reported as a set of key/value pairs. This data is followed by a possibly empty set of TUNNEL specifications associated with this node.

The key/value pairs are:

```
      authentication --- value is the IPsec authentication types that
                           are supported by this node separated by
                         whitespace.  Examples: 'none' or 'none md5'
      class ------------ function of this node in this overlay, value
                           is either host or router
      dns_name --------- value is the base DNS name of this node
                           in principal, this is optional
      dynamic_routing -- value is either 'yes' or 'no'

      encryption ------- value is the IPsec authentication types that
                           are supported by this node separated by
                         whitespace.  Examples: 'none' or 'none des'
      interfaces ------- unsigned number of interfaces (tunnels)
                           currently used by or available to this node
      ip_address ------- value is the base IPv4 or IPv6 numeric address
                           for this node
       max_interfaces --- max. number of interfaces (tunnels) supported
                           by this node
      max_overlays ----- max. number of overlays supported by this node

      os --------------- value is the operating system of this node

      overlays --------- current number of overlays at this node

      release ---------- release version of Xbone running on this node
                           format is 'int.alphanumeric'
      status ----------- value is the status of this node in this overlay,
                           either in or out
      tunnel_count ----- number of tunnels in use by this node
```

Not all key/value pairs are present in every NODE specification. Some key/value interpretations depend upon the enclosing message type.

TUNNEL Rule

```
TUNNEL:  (  tunnel  CRITERION(s) )

Each TUNNEL describes the state of a particular tunnel in an overlay.
This state is reported as a set of key/value pairs.

They key/value are:

          local_ip_address --- value is a numeric form IPv4 or IPv6 address
          remote_ip_address -- value is a numeric form IPv4 or IPv6 address
          status ------------- value is either up or down
```

XOL

Xbone Overlay Language Specification

Overview

The goal of the language is to unburden the user from a requirement to specify an overlay by laboriously listing each router, host, their interfaces and the links between them. Users should be able to specify an overlay with varying degree of precision as fits their needs. The Overlay Managers (OMs) will then choose a specific overlay configuration that matches the user requirements.

This is accomplished in the language by allowing overlays to be specified with degrees of ambiguity. An example of a somewhat ambiguous specification is to list the routers and their interconnecting links, but leave unspecified which host was connected to which router. The language must also allow the fully complete specification of an overlay.

The job of the OMs is to analyze the overlay program, generate a resource requirements list and assemble a set of routers, hosts and interfaces to fill that list. The OMs then transform the overlay program from a potentially imprecise topology specification into a particular overlay network.

This transformation progresses by pinning particular routers, hosts and interfaces to their generic counterparts in the program. The user controls the pinning undertaken by the OMs to the extent that ambiguity is present in the initial overlay program.

Overlay Components

An overlay contains: interfaces, links, routers, hosts and a network list. A host has one interface, while routers have one or more interfaces. Each of these components may also be associated with certain user-specified properties. Links are used to interconnect interfaces.

Interfaces, hosts, routers and links have identifications (names) that are unique within the overlay. Each overlay also has a name and implicitly a boundary. An overlay boundary may contain one or more interfaces that it exports. Thus, an overlay is similar to a router in external appearance and may itself be a component within another overlay. Components within an overlay are never exported outside their overlay. Interior components are logically invisible outside an overay.

The interior topology of an overlay is represented by a network list. Similar to a graph, the hosts and routers of the overlay are its nodes, while links are its edges. The degree of a node corresponds to the number of interfaces associated with the overlay, router or host that it represents.

Links must always be pinned at one end to a specific interface of one router or host. The other end of a link may be unspecified to varying degrees, in which case it is said to float. For example, assume one end of a link is pinned to the interface of a host. The other must ultimately connect to the interface of some router, but the user could leave the pinning to the choice of the OM by not specifying which router or interface.

Conceptually, hosts and routers may be thought of as atoms in a molecular model. Hosts or routers have links with floating interfaces (bonding points) emerging from them that can be pinned (bonded) to another appropriate link and interface. The OM transforms an ambiguous overlay specification to a specific overlay instance (molecule) by appropriately bonding the floating interfaces to one another. If the OM has insufficient routers, hosts or interfaces to match the requested overlay, it fails to build the overlay network. If presented with more then the needed number of components, those unneeded will be left out of the resulting network instance.

Object Identifiers

XOL defines overlay systems. An overlay system is comprised of resources: overlays, hosts, routers, links and interfaces. Within an individual overlay hosts, routers, and links are uniquely named.

Interfaces are physically bound to a particular overlay, host or router and are not referred to in isolation. Therefore, interface names need be unique only in the context of their associated overlay, router or host.

If more than one overlay is defined in the system, each overlay name must be unique.

Class Identifiers

Certain resources in XOL may also be denoted as a set. To determine to which type of resource a set identification refers, a suffix is appended.

The resource class suffixes are:

Class	Suffix
Router	.R
Host	.H
Interface	.I

Thus, within the overlay the expression "*.H" refers to any host and "*.R" to any router. The expression "*.I" refers to any interface of the overlay, router or host with which it is contextually associated.

Links

Overlays, routers and hosts possess interfaces to which links are pinned (attached). The interface at the end of the link may be described as floating or pinned. A pinned interface is one that is associated with the interface of a particular overlay, router or host.

A link has the syntactic form:

```
<link> ::=    <link_name> <lcl_end_association> <rem_end_association>
```

Links have both local and remote end-associations. A remote end-association specifies the resource and resource interface to which a link end is or may be attached.

The interface of the host or router is always pinned to the local end of a link that emerges from it. The remote end of such a link may be either pinned or floating. A link with a floating remote end-association has a set of possible end-associations.

Floating link ends can be associated with some other matching floating link end, thus creating a single fully-pinned link. A fully-pinned link constitutes a one-to-one, symmetric association between two interfaces. A fully-pinned link connects a router interface to another router interface, a router interface to an overlay interface or a router interface to a host interface.

The procedure that creates fully-pinned links progresses as the OM turns an ambiguous overlay specification into a particular overlay instance.

End-Associations

An end-association consists of two fields. The leftmost field can name a specific overlay, router or host resource. It can also float, specifying one of those three resource classes. The rightmost field specifies either a specific interface or floats, specifying an interface class.

For example:

```
      *.R:*.I       --- some interface of some router

      tom:*.I       --- some interface of tom, tom specifying a
                          particular router host or overlay

      tom:tom0      --- the tom0 interface of tom, tom specifying a
                          particular router host or overlay
```

A field is considered pinned if it names a specific resource (an overlay, router, host or interface). A pinned end-association contains two pinned fields.

A floating remote end-association contains at least one regular expression pattern (PERL regular expression) that can match one or more specific resources. For example, the end-association *.H:*.I would match any host within the overlay and any of their interfaces.

Implicit Link Restrictions

When the leftost field of an end-association floats, the rightmost fieldmust match any interface. Thus, specifying any interface of a specificoverlay, router or host is legal, as in tom:*.I, but the specification *.R:ed0 is illegal.

If the local end-association of a link specifies a host, the remote end-association, whether floating or pinned, must specify a router or router class end-association.

A particular link name may appear in only one netlist rule in an particular overlay topology specification.

Intra-Overlay Resolution Phase

A consistent network definition requires that all intra-overlay host and router end-associations specified by the topology become fully pinned during the overlay resolution phase carried out by this overlay's controlling OM.

An additional property of intra-overlay resolution is that multiple logical routers and hosts may be overloaded into fewer physical hosts or routers. It is the responsibility of the controlling OM to ensure that sufficient virtual interfaces are available in the physical host when such overloading occurs.

Intra-Overlay Resolution Phase

Any floating overlay class end-associations become pinned during the inter-overlay resolution phase carried out between the OMs controlling the concerned overlays.

XOL Syntax

The Xbone Overlay language, is specified in an extended BNF that is accepted by the RecDescent compiler-generator package, which is provided as part of the Xbone distribution. The output of the API compilation is a loadable PERL data structure that captures the overlay specification.

A short discussion of the extended BNF conventions allowed by RecDescent is presented, followed by a keyword list and the grammar itself. Compile-time semantics are then discussed.

Finally, a detailed description of the compiled data structure is presented. This will be of interest primarily to those maintaining or extending the XBONE system.

```
  XOL_PROGRAM:  '('  'xol'  OVERLAY(1..)  ')'

  OVERLAY:  '('  'overlay'  NET_PART(s)  ')'

  NET_PART:  NET_NAME
          |  NET_ADDRESSTYPE
          |  NET_SECURITY
          |  NET_SERVICES
          |  NET_TOPO
          |  INTERFACES

  NET_NAME:  '('  'name'  ARGSTRING  ')'

  NET_ADDRESSTYPE: '('  'addresstype'  ADDRESSTYPE  ')'

  NET_SECURITY:  '('  'security'  SEC_CLAUSE(s)  ')'

  SEC_CLAUSE:  NET_CREATOR
            |  NET_ENCR
            |  NET_AUTH
            |  NET_EMAIL
           |  CRITERION

  NET_CREATOR:  '('  'creator'  ARGSTRING  ')'

  NET_EMAIL:  '('  'email'  ARGSTRING  ')'

  NET_ENCR:  '('  'encryption'  ENCR_ALG  ')'

  NET_AUTH:  '('  'authentication'  AUTH_ALG  ')'

  NET_SERVICES:  '('  'services'  NET_SERVICE(2..)  ')'

  NET_SERVICE:  NET_MGR
             |  NET_ADDRSVR
             |  NET_DOMAINSVR
             |  NET_MONITOR
             |  NET_APPLICATIONS

  NET_MGR:  '('  'manager'  NETADDR  PORTSPEC(?)  ')'

  NET_ADDRSVR:  '('  'address_server'  NETADDR  PORTSPEC(?)  ')'
```

```
NET_DOMAINSVR:  '('  'name_server'  NETADDR  PORTSPEC(?)  ')'

NET_MONITOR:  '('  'monitor'  ARGSTRING  ARGSTRING(s?)  ')'

NET_APPLICATIONS:  '('  'applications'  NET_APPLICATION(s)  ')'

NET_APPLICATION:  '('  'application'  ARGSTRING  ARGSTRING(s?)  ')'

PORTSPEC:  'port'  ARGSTRING

NET_TOPO:  '('  'topology'  CANON_SPEC(s)  ')'

CANON_SPEC:  '('  'routers'  RTR_SPEC(s)  ')'
          |  '('  'links'  LNK_SPEC(s)  ')'
          |  '('  'hosts'  HST_SPEC(s)  ')'
          |  '('  'netlist'  TOPO_TRIPLE(s)  ')'

TOPO_TRIPLE:  '('  OBJECT_NAME  LCL_ASSOC  REM_ASSOC  ')'

RTR_SPEC:  '('  OBJECT_NAME  INTERFACES  NODE_CRITERION(s?)  ')'

HST_SPEC:  '('  OBJECT_NAME  INTERFACES  NODE_CRITERION(s?)  ')'

LNK_SPEC:  '('  OBJECT_NAME  NODE_CRITERION(s?)  ')'

INTERFACES:  '('  'interfaces'  INTERFACE(s)  ')'

INTERFACE:  '('  OBJECT_NAME  NODE_CRITERION(s?)  ')'

LCL_ASSOC:  OBJECT_NAME  ':'  OBJECT_NAME

REM_ASSOC:  OBJECT_SPEC  ':'  IFACE_SPEC

HOST_SPEC:  OBJECT_NAME
         |  HOST_CLASS

OBJECT_SPEC:  OBJECT_NAME
           |  OBJECT_CLASS

IFACE_SPEC:  OBJECT_NAME
          |  IFACE_CLASS

CRITERION:  '('  ARGSTRING ARGSTRING  ')'

NETADDR:  IPV4_ADDR
       |  IPV6_ADDR
       |  DNS_NAME

DNS_NAME: ARGSTRING          ... (must conform to DNS name syntax)

ADDRESSTYPE:  'ipv4'

ENCR_ALG:  'des'
        |  '3des'
        |  'none'

AUTH_ALG:  'md5'
        |  'sha1'
        |  'none'

IPV4_ADDR:  /[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/
```

```
IPV6_ADDR:  ARGSTRING       ... (must conform to IPv6 syntax)

ARGSTRING:  /"[^"()]+"/
         |  /'[^'()]+'/
         |  /[^()\s]+/

OBJECT_NAME:  /[\._A-Za-z0-9-]+/
           |  /"[\._A-Za-z0-9-]+"/
           |  /'[\._A-Za-z0-9-]+'/

HOST_CLASS:  /\*\.H/

ROUTER_CLASS:  /\*\.R/

OVERLAY_CLASS:  /\*\.O/

IFACE_CLASS:  /\*\.I/
```

Example Overlay Specifications

An overlay program can specify its topology directly. In the following example of a small line network, each link and interface assignment to link is explicitly or implicitly defined.

The description specifies each resource needed for the overlay: hosts, routers, their virtual interfaces and virtual links that could interconnect them. Links are explicitly connected at their local end to the interface of a specific router or host. Their remote ends may be connected to the interface of some other router, host or overlay.

The resource names are merely reference placeholders. They bear no particular relationship to any actual hostnames. They could equally well be integers. The programmer is free to use names suggestively as an aid to correct overlay specification.

This program also illustrates that the specification allows the programmer to attach properties, that are known to the OM, to any resource. Here, for example, the operating system of each host is "freebsd", for routers "linux", and no links have properties attached to them.

Only three links are needed to specify this overlay in isolation. An additional link is provided from router able to the overlay's exterior interface TL0, which allows the overlay to be seen externally. Such links and interfaces are needed when multiple overlays are being specified by the program.

```
( xol
 ( overlay
  (name "TESTLINE")
  ( security
    (creator "GGFranklin") (email 'ggf@isi.edu')
    (encryption none)
  )
  ( services
    ( manager 128.9.160.237 port 123 )
    (name_server cnn)
    (monitor magic_fingers.exe -ping -10 )
    ( address_server "moo.isi.edu" )
    ( name_server moo.isi.edu )
  )
  (interfaces   (TL0) )
  ( topology
      ( hosts
       ( dick
         (interfaces   (dick0) )
          (os "freebsd")
       )
       ( tom
            (interfaces   (tom0)  )
            (os "freebsd")
       )
      )
      ( links
       (link0)
       (link1)
       (link2)
       (link3)
      )
      ( routers
       ( able
         (interfaces   (able0)  (able1)  (able2))
            (os "linux")
       )
       ( baker
         (interfaces   (baker0) (baker1) )
         (os "linux")
       )
      )
      ( netlist
       (link0  tom:tom0      able:able0)
       (link1  able:able1    baker:baker0)
       (link2  baker:baker1  dick:dick0)
       (link3  able:able2    TESTLINE:TL0)
      )
  )
 )
)

[end of program example]
```

The previous overlay fully pinned (completely specified) the remote ends of all its links. That is not grammatically necessary. The remote end of links can connect to resource classes, rather than to specific elements in a class. However, see the Appendix below for implementation restrictions imposed by limitations of the current Xbone release.

In the program below, hosts tom and dick can connect to any available router interface, indicated in the netlist via "*.R:*.I." Correspondingly, each router has one link and port that can connect to any host interface. Finally, router able can connect via its able2 interface to any exterior interface of the TESTLINE overlay.

One restriction is that if the remote end host, router or overlay is class specified, so must be the interface. The remote specification "*.R:bob1" is illegal.

```
( xol
 ( overlay
  (name "TESTLINE")
  ( security
    (creator "GGFranklin") (email 'ggf@isi.edu')
    (encryption none)
  )
  ( services
    ( manager 128.9.160.237 port 123 )
    (name_server cnn)
    (monitor magic_fingers.exe -ping -10 )
    ( address_server "moo.isi.edu" )
    ( name_server moo.isi.edu )
  )
  (interfaces   (TL0) )
  ( topology
      ( hosts
       ( dick
         (interfaces   (dick0) )
           (os "freebsd")
       )
       ( tom
            (interfaces   (tom0)  )
            (os "freebsd")
       )
      )
      ( links
       (link0)
       (link1)
       (link2)
       (link3)
       (link4)
       (link5)
      )
      ( routers
       ( able
         (interfaces   (able0)  (able1)  (able2))
            (os "linux")
       )
       ( baker
         (interfaces   (baker0) (baker1) )
         (os "linux")
       )
      )
      ( netlist
       (link0  tom:tom0      *.R:*.I)
       (link1  dick:dick0    *.R:*.I)
       (link2  able:able0    *.H:*.I)
       (link3  baker:baker0  *.H:*.I)
       (link4  baker:baker1  able:able1)
       (link5  able:able2    TESTLINE:*.I)
      )
  )
 )
)

[end of program example]
```

Security Convention

The single AUTH_ALG and ENCR_ALG is applied across the entire network to all its links in both the forward and backward directions. If AUTH_ALG or ENCR_ALG is "none", no authentication or encryption algorithm respectively is applied to the links of the network.

Variable Constraints

A DNS_NAME is an ASCII string that is compatible with the Domain Name System (DNS) specification of a fully-specified host name.

A NETADDR is an ASCII numeric string that expresses an IPv4 address in dotted decimal format or in DNS_NAME form. The string passed is checked first for whether or not it has legal numeric address syntax. If it does not it is assumed to be a DNS name. DNS names are also checked for syntactic legality.

A PORTSPEC is a 16-bit non-negative integer. There are system restrictions as to which actual numbers may be used. Contact your network system administrator.

A MONPROG is the name of a program known to the OM. This program monitors the health of the overlay network created by the OM. Interaction model between the program and the OM is yet to be defined.

The MONPROG_PARAMS is a sequence of staticly defined argument strings used when the MONPROG is started.

Compiled Overlay Data Structure

The compiled output of an XOL program is a loadable PERL data structure. In what follows, the major linguistic rules are named along with their corresponding compiled data structures.

Although it is not generally made clear in PERL documentation, lists and hashes contain only scalar objects. When the child object to be contained is non-scalar (either a list or hash), the parent contains a referenceto the child object. Thus, in a list of lists the parent contains a list of scalar references to lists.

In the following description it should be understood that both lists andhashes that contain compound objects actually contain references to thoseobjects. Thus, it will be said that "x contains a list of hashes",rather than the more cumbersome "x contains a list of references to hashes",

In the following description the only values discussed are those returned when no syntactic or semantic error has been detected. Any such error aborts the compilation.

```
SYSTEM:  '('  'system'  OVERLAY_SYSTEM  ')'  OVERLAY(1..)  END_OF_FILE
```

Returns a reference to a hash that contains the compiled representation for the entire system of overlays. The defined keys are: system and overlays.

The value of system is the OVERLAY_SYSTEM hash. The value of overlays is a list of one or more component OVERLAY hashes.

```
OVERLAY_SYSTEM:  '('  'system'  '('  NAME  SYSTEM_VERSION  ')'  ')'
```

Returns a reference to a hash. The defined keys are: system and version. Their values are the string representation of the XBONE overlay system program and its release version, respectively.

```
OVERLAY:  '('  'overlay'  NET_PART(s)  ')'
```

This is a hash that captures the structure of an individual overlay. A reference to a hash is returned. The defined keys are: net_name, net_addresstype, net_security, net_services, net_interfaces and net_topology. Only one of each of these is allowed in each overlay specification.

The net_name value is a string name for the network. The net_addresstype is optionally specified. If not specified, the value "ipv4" is provided. The net_security value is a NET_SECURITY hash (see below). The net_services value is a NET_SERVICES hash (see below). The net_interfaces value is an INTERFACES hash (see below). The net_topology value is a NET_TOPOLOGY hash (see below).

```
NET_SECURITY:  '('  'security'  SEC_CLAUSE(s)  ')'
     Returns a reference to a hash.  The defined keys are: creator, email,
     encryption and authentication.  The creator and email keys have values that
     are the network creator's personal name and their email address, respectively.
     Both encryption and authentication are optional.  If present, they are the
     names of an encryption or authentication algorithm, respectively.  These
     algorithms must be known to the XBONE system.  If not present or needed, the
     value "none" is provided.


NET_SERVICES:  '('  'services'  NET_SERVICE(2..)  ')'
     Returns a reference to a hash.  The defined keys are: net_manager,
     net_domainsvr, net_addrsvr, net_applications and net_monitor.  The values for
     net_manager, net_domainsvr and net_applications are lists of NET_MGR
     NET_DOMAINSVR and NET_APPLICATION hashes.  The value for net_addrsvr and
     net_monitor are NET_ADDRSVR and NET_MONITOR hashes.


NET_MGR:  '('  'manager'  NETADDR  PORTSPEC(?)  ')'
     Returns a reference to a hash.  The defined keys are: manager, and
     optionally, port.  The manager value is the NETADDR passed and the port value
     if present is that of the PORTSPEC.

     Each XBONE overlay network must have one or more overlay manager servers
     (OMs).  This OM resides at the specified address and port number.


NET_ADDRSVR:  '('  'address_server'  NETADDR  PORTSPEC(?)  ')'
     Returns a reference to a hash.  The defined keys are: address_server, and
     optionally, port.  The address_server value is the NETADDR passed and the port
     value if present is that of the PORTSPEC.

     Each XBONE overlay network must have one or more network address servers.
     This address server resides at the specified address and port number.


NET_DOMAINSVR:  '('  'name_server'  NETADDR  PORTSPEC(?)  ')'
     Returns a reference to a hash.  The defined keys are: name_server, and
     optionally, port.  The name_server value is the NETADDR passed and the port
     value if present is that of the PORTSPEC.

     Each XBONE overlay network can have one or more Domain Name System (DNS)
     servers.  This DNS server resides at the specified address and port number.


NET_MONITOR:  '('  'monitor'  MONPROG  MONPROG_PARAM(s?)  ')'
     Returns a reference to a hash.  The defined keys are: monitor and params.
     The monitor value is a string that names a monitoring program known to the
     XBONE system.

     The params value is a list of zero or more strings.

     Currently, there is no monitoring program known to the XBONE system.

NET_APPLICATIONS:  '('  'applications'  NET_APPLICATION(?)  ')'
     Returns a reference to a hash.  The defined key is: applications.
     Its value is a reference to a list of one or more NET_APPLICATION objects.
```

```
NET_APPLICATION:  '('  'application'  ARGSTRING  ARGSTRING(s?)  ')'
     Returns a reference to a hash.  The defined keys are: program and params.
     The first ARGSTRING is taken to be the program, the remainder for the params.
     The program value is a string that names an application program known to the'
     XBONE system.  Two programs are currently known:  "URL" and "Dummynet".

     The params value is a list of zero or more strings.

     The "URL" program has one argument in its params list.  That argument is a
     string taken to be a URL specification of the application program to be run.

     The "Dummynet" program has up to six arguments in its params list.  They are
     strings of the form "name=value".  The value's are defined by the DUMMYNET
     software and documented elsewhere.

            "Delay=value"
            "Bandwidth=value"
            "Bandwidth_unit=value"
            "Queue=value"
            "Queue_unit=value"
            "Loss_rate=value"


INTERFACES:  '('  'interfaces'  INTERFACE(s)  ')'
     Returns a reference to a hash.  The keys are the interface names defined.
     The value is a reference to a hash of key/value pairs that are associated
     with this interface name.

     The interface names defined here represent the external interfaces to the
     overlay being defined.


NET_TOPO:  '('  'topology'  CANON_SPEC(s)  ')'
     Returns a reference to a hash.  This creates the hash that contains
     the canonical representation of the topology of the overlay.  This hash
     contains four keys: routers, hosts, links and netlist.

     The routers key value is a reference to a hash of internal router names.
     In the routers hash, each key is the unique name of a router.  The value
     associated with each such key is itself the hash of CRITERION
     (key, value) pairs specified in the accompanying RTR_SPEC (see below).

     The hosts key value is a reference to a hash of internal host names.  In
     the hosts hash, each key is the unique name of a host.  The value
     associated with each such key is itself the hash of CRITERION
     (key, value) pairs specified in the accompanying HST_SPEC (see below).

     NOTE: The internal names for each router, host and the enclosing
           overlay name form a single set of unique names.  In other
           words, a router's internal name cannot be the same as that
           of a host or that of the enclosing overlay.

     The links key value is a reference to a hash of internal link names.
     In the links hash, each key is the unique name of a link.  The value
     associated with each such key is itself the hash of CRITERION
     (key, value) pairs specified in the accompanying LNK_SPEC (see below).

     The netlist key value is a reference to a list of TOPO_TRIPLE three-element
     lists (see below).
```

```
TOPO_TRIPLE:   '('  OBJECT_NAME  LCL_ASSOC  REM_ASSOC  ')'
     Returns a reference to a hahlist.  The defined keys are: link, lcl_object,
     lcl_iface, rem_object and rem_iface.

     Every host and router must have a named link associated with each of its
     named virtual interfaces.  This information is captured by naming a link
     and pinning one end of it in the netlist rule specification.  Thus, the
     OBJECT_NAME names a link, while the LCL_ASSOC specifies both a router or
     host and one of the virtual interfaces, as in "bob:bob1", where bob is the
     internal name of a router or host and bob1 the name of one of its
     interfaces.

     The REM_ASSOC or remote end of a named link can either be completely
     pinned by specifying another overlay, host or router and one of their
     interfaces.  However, to provide the freedom to specify a family of
     possible topologies for an overlay, the REM_ASSOC can also specify a
     class of possible hosts, routers and interfaces, as in "*.R:*.I".

     The link value in the TOPO_TRIPLE hash is the name of a link
     that must be defined in the CANONICAL_TOPO links hash.  The LCL_ASSOC
     is split into the lcl_object and lcl_iface (key, value) pairs, where the
     value is the router or host internal name and internal interface name.
     The REM_ASSOC is split into the rem_object and rem_iface (key, value) pairs,
     where the value is the overlay, router or host internal name and internal
     interface name, or a class specifier.


RTR_SPEC:  '('  OBJECT_NAME  INTERFACES  CRITERION(s?)  ')'

HST_SPEC:  '('  OBJECT_NAME  INTERFACES  CRITERION(s?)  ')'

LNK_SPEC:  '('  OBJECT_NAME  CRITERION(s?)  ')'
     Returns a reference to a two-element list.  The first element is the internal
     reference name given to a router, host or link.  The second element is a
     reference to a hash that contains the (key,value) pairs specified as this
     links properties.  If no properties are specified, the second element will
     point to an empty hash.


CRITERION:  '('  ARGSTRING ARGSTRING  ')'
     Returns a reference to a hash.  The defined keys are: type and value,
     take on the values of the first and second ARGSTRING respectively.
```

Appendix

Xbone Release 2.0 Implementation Constraints

The XOL grammar provides a number of features that the current release of the Xbone does not support. These are listed below.

Lack of Two-Phase Implementation

The "host_choice" and "host_choice_reply" messages exist to allow two-phase commitment. The OM sends a "host_choice" message out its API to the overlay builder. The builder then picks those hosts that are to participate and returns a "host_choice_reply" response to the OM, which then builds the overlay.

The overlay builder can leave the choice of hosts entirely to the OM. The 'two_phase' option field of the "create_overlay" command controls this. If the value of 'two_phase' is 'yes', the two-phase commit message exchange occurs. Otherwise, the value must be 'no'.

In this 2.alpha release, two-phase commitment is not supported. The "host_choice" and "host_choice_reply" messages are not currently implemented. Thus, the 'two_phase' option field of the "create_overlay" message must be 'no'.

Overlay Specification

The Xbone GUI currently 'writes' the XOL programs needed to create overlays for star, ring and line topologies only. Other topologies are not supported.

The XOL programs that the GUI creates must be non-ambiguous. That is, they must fully pin (completely specify) the remote ends of all links. The OM does not currently support any floating (wildcard specification) of remote links.

OMs currently support only single-network overlays. Thus, in the XOL_PROGRAM rule, only one OVERLAY specification is suported. The INTERFACES part of the OVERLAY specification must be present, but is unused.

Address Specification

While the grammar supports both IPv6 and IPv4 addressed, at this time only IPv4 addresses are supported.

MONPROG Specification

The MONPROG feature of the grammar is not supported.

Example: Complete API/XOL Program

This is the textual representation of a 'create_overlay' API command. The create_overlay command contains an XOL program specification for an overlay.

```
( xbone 1.5 2.0
   (create_overlay
       (auth_type  x509)  (creator_name  'Yu-Shun Wang')
       (creator_email  yushunwa@isi.edu)  (two_phase  no)
       (dns  yes)  (hosts  2) (host_os  kame) (authentication  sha1)
       (encryption  3des)  (dynamic_routing  no)
       (overlay_name  test-1.xbone.net)  (routers  2)
       (router_os  kame)  (search_radius  5)  (topology  linear)
       (user_id yushunwa@isi.edu)
       (program
          (xol
             (overlay
                (name test-1.xbone.net)
                (addresstype ipv4 )
                (security
                    (creator_name 'Yu-Shun Wang')
                    (creator_email yushunwa@isi.edu)
                    (authentication sha1)
                    (encryption 3des)
                )
                (services
                    (manager www.xbone.net)
                    (name_server rum.isi.edu)
                )
                (interfaces (test-1.xbone.net0) )
                (topology
                   (hosts
                       (_hst_0_ (interfaces (_hst_0_0)) (os  kame) )
                       (_hst_1_ (interfaces (_hst_1_0)) (os  kame) )
                   )
                   (routers
                       (_rtr_0_ (interfaces (_rtr_0_0)(_rtr_0_1)) (os  kame) )
                       (_rtr_1_ (interfaces (_rtr_1_0)(_rtr_1_1)) (os  kame) )
                   )
                   (links
                       (link0)
                       (link1)
                       (link2)
                   )
                   (netlist
                       (link0 _rtr_0_:_rtr_0_1 _rtr_1_:_rtr_1_1)
                       (link1 _hst_0_:_hst_0_0 _rtr_0_:_rtr_0_0)
                       (link2 _hst_1_:_hst_1_0 _rtr_1_:_rtr_1_0)
                   )
             )
           )
         )
       )
   )
)
XboneEOC
```