Section E.2.3

Hesiod Name Service

by Stephen P. Dyer and Felix S. Hsu

Hesiod: 8th century B.C. Greek farmer and poet. His Theogony *was the first work that systematically recorded the names of the gods and the myths surrounding them.*

Status

This document describes the technical plan for Hesiod, the Athena Name Service. As of this writing, the initial implementation of the Hesiod name resolution library has been integrated into several production network applications (remote disk, remote file, printer service, login service, and post office service) in anticipation of campus-wide deployment in Fall, 1987.

1. Purpose

Hesiod, the Athena name server, provides access to services in a distributed environment, allowing centralized administration of information via the Athena Service Management System (SMS) or other sources of authoritative information. More specifically, it replaces databases which heretofore have had to be duplicated on each workstation and timesharing machine (rvdtab, clustertab, printcap, services) and is a flexible mechanism to provide new information to services as the need arises. It also allows subsidiary organizational entities under Athena to manage their own information. The Athena Name Server is NOT the Athena SMS, although it may obtain much of its information from the SMS. In brief, it provides a content-addressible memory where certain strings can be mapped to others depending on the type of query made. The name server has no knowledge about the data it stores, meaning that queries and responses are simple key/content interactions; ad-hoc, generalized queries are not supported.

2. Scope

The Hesiod name service is logically distinct from the Internet name service which provides host name and internet address lookup, as well as other information related to mail handling. However, in the present implementation, both services are provided by a version of BIND, the Berkeley Internet Name Daemon, with Hesiod provided by a layer of routines which invoke BIND. This document discusses only the Hesiod name service and the manner in which BIND supports it. A detailed description of the Internet Name Server and the BIND implementation in particular can be found in ARPANET Request For Comments numbers RFC882, RFC883, and RFC973, the Bind Operations Guide and UCBerkeley reports UCB/CSD 84/177 and 84/182.

3. Hesiod Names

Hesiod is layered on top of BIND, and hence may use a naming syntax which is not identical to BIND if greater expressiveness is needed, provided that the names which are ultimately passed to the BIND package are valid BIND strings. There are two problems when relying on standard Internet domain notation to refer to objects named by Hesiod. First, the desire to have objects whose relative names contain the '.' character conflicts with Internet domain notation, where a name which contains any '.' is considered fully resolved. Second, the standard BIND implementation of the Internet domain server has no provision for deciding the proper domain suffix to use when resolving a relative name, since there was only one possibility when handling Internet queries. In considering these problems, we came up with the following scheme:

A name given to the Hesiod name server for resolution looks like:

```
HESIODNAME => LHS
HESIODNAME => LHS@RHS
LHS        => [Any ASCII character, except NUL and '@']*
                      { 0 or more characters from this alphabet }
RHS        => [Any ASCII character, except NUL and '@']+
                      { 1 or more characters from this alphabet }
```

In other words, a hesiodname consists of

```
[LHS][@RHS]
```

where either [LHS] or [@RHS] need not be present.

The LHS of a Hesiod name is *uninterpreted*; although it may be modified according to the rules described by the information in */etc/hesiod.conf* (see below), it is not itself a domain name.

We define a set of routines known as the Hesiod library which takes a Hesiod name and a user-supplied key, known as a *HesiodNameType*, converts it to a fully-qualified domain name, calls the BIND library, and returns the result to the original caller. In the current design, translation of the result is null, but the option is available in future refinements to do some translation if it should be required. The *HesiodNameType* is a well-known string which is provided by an application which uses the Hesiod library. It is used directly in the expansion of a Hesiod name to a BIND name (see below) without further indirection or translation. A new *HesiodNameType* comes into existence simply by being used by an application; no libraries or configuration files need to be modified. Naturally, there has to be appropriate data stored by the name server which is associated with that *HesiodNameType*.

To provide an example, one of the routines in the Hesiod library takes a Hesiod name and returns a fully-qualified name to be handed to BIND:

```
char *
hes_to_bind(HesiodName, HesiodNameType)
char *HesiodName, *HesiodNameType;
```

The *HesiodNameType* identifies the query to make to BIND, and the proper expansion rules to use with the LHS and RHS of the name. This would be chosen by the application, and could be application-specific.

Thus, the following are valid Hesiod names:

```
14.21
default-printer
default-printer@SIPB
@heracles
@heracles.MIT.edu
finger-server@Berkeley.MIT.edu
```

/etc/hesiod.conf contains two tables specifying the treatment of LHS and RHS components of a Hesiod name. In the translation of a Hesiod name to a valid BIND name, the LHS is expanded by concatenating together the Hesiod name, the separator '.', the HesiodNameType, and the LHS entry found in */etc/hesiod.conf*. If the RHS is null, the RHS entry in */etc/hesiod.conf* is used. If the RHS is a fully qualified domain name already, it is used directly. Else, if a RHS is present, it is used as a Hesiod name for further resolution against the HesiodNameType, *rhs-extension*. If this query succeeds, the first reply is used as the RHS, otherwise an error is returned. The fully-expanded LHS and RHS are then concatenated together, separated by a '.', and this value is passed to BIND.

With the definition of */etc/hesiod.conf* given below, a call to hes_to_bind("e40", "printer") would produce a LHS of "e40.printer.ns" and a RHS of ".Athena.MIT.edu", and the resulting BIND name would be "e40.printer.ns.Athena.MIT.edu".

The following is a typical copy of /etc/hesiod.conf:

```
#file /etc/hesiod.conf
#comment lines begin with a '#' in column 1
#LHS table
lhs = .ns
#RHS table
rhs = .Athena.MIT.edu
```

In C pseudo-code, we would have the following productions:

```
hes_to_bind(14.21, filesys)
                          => 14.21.filesys.ns.Athena.MIT.edu
hes_to_bind(e40, printer)
                            => e40.printer.ns.Athena.MIT.edu
hes_to_bind(SIPB, rhs-extension)
                       => SIPB.rhs-extension.ns.Athena.MIT.edu
hes_to_bind(default@SIPB, printer)
                            => default.printer.ns.SIPB.MIT.edu
(assumes that previous production resolved to "SIPB.MIT.edu")
hes_to_bind(finger-server@Berkeley.EDU, serviceloc)
                         => finger-server.sloc.ns.Berkeley.EDU
```

4. Hesiod and BIND

4.1. BIND requirements

A version of BIND which supports the locally added C_HESIOD query class and T_UNSPECA query type is required to support the Hesiod name service. The standard Athena release of BIND has been modified to support this. The changes will also be forwarded to Berkeley to be included as the standard BIND release. We have asked Paul Mockapetris to make C_HESIOD and T_UNSPECA official so that other sites might be able to use Hesiod, and he seems amenable to this.

4.2. BIND's view of the Hesiod Name Space

The name space within Project Athena is structured as follows: workstations and timesharing machines will be moved from the MIT.edu domain to the domain, Athena.MIT.edu. All internet name server queries, e.g., host-to-address lookup, are made relative to the Athena.MIT.edu domain. Hesiod name server queries will be made relative to the NS.Athena.MIT.edu domain. This arrangement allows internet address assignment for Athena.MIT.edu to continue to be handled by MIT Telecommunications with its own authoritative name server and database, while the data for the Athena name service can be administered by Athena operations. However, other administrative entities may choose other naming arrangements, for example, the DU fraternity could run its own name service for local queries with a domain suffix of DU.Athena.MIT.edu. The current implementation of BIND and Hesiod allows any administrative group to run a name server which is authoritative for its own domain.

During the design phase, a number of approaches were considered; a "flat" name space where all objects could be named by an atomic string, and a hierarchical name space, roughly reflecting the organization of clusters out in the field, where a name consisted of a hierarchy of domain-style names. Both cases would ultimately terminate in a qualifying domain string such as "NS.Athena.MIT.edu". For example, a system library might be named "sys.rt.e40.NS.Athena.MIT.edu", where "sys" reflected the fact that it was a system RVD pack, "rt" indicated that it was an RVD pack for the RT, found in cluster "e40". Although the hierarchy appears to specify a series of separate domains, the presence of a "." need not indicate a separate zone of authority, This scheme allowed for simple composition of names at the expense of a somewhat more complicated naming scheme. The current technical plan assumes a flat name space.

4.3. BIND Queries

Every BIND query and resource record is qualified by a *class* and a *type*. In the scheme described in RFC882 and RFC883, the *class* is intended to loosely identify the contents of the resource record returned by the name server. The two "official" classes recognized by the NIC are "internet" and "chaos", with small integer values 1 and 3, respectively. The class "any", value 255, is used as a wild-card in a query, indicating that the class of a resource record does not matter. To this, we add the class "hesiod", value 4. Resource records for Hesiod names returned by BIND are tagged with this class. The *type* specifies the kind of data in a resource record. A number of standard types have been defined by RFC883, and the BIND implementors have defined several additional types. Hesiod uses a newly-defined type, T_UNSPECA, which is a generic type class indicating that a resource record contains 'unspecified ASCII data' presumably of interest only to the invoking application and not meaningful to BIND.

5. Athena Client Applications of Hesiod

The five applications which will immediately use the Hesiod name server are the Hesiod library itself, RVD utilities, MDQS-derived line printer utilities, a replacement for the passwd server used by *Toehold* and modifications to the *Kerberos* libraries to query the name server to locate one's Kerberos server. There are many other potential applications, some of which are sketched out in section 6.5, "Proposed Additional Queries". For each of these four applications, we identify the form of the queries and the data returned by Hesiod, what changes needed to be made to the application, and where the authoritative data will come from.

5.1. Cluster Information

Certain services, such as remote system libraries and printers, are best though of a belonging to a 'service cluster'. The *cluster* query returns the names of the service clusters for all such services, which at present consist of the *Hesiod name keys*, *syslib*, *userlib*, *printer* and *kerberos*. Note that the service cluster name is not necessarily a host name, but a Hesiod name which may be resolved by the application later. The service name which identifies each service cluster is a literal string representing an environment variable which will be assigned the service cluster as a value.

```
General...

BIND name:      workstationname.cluster.ns.Athena.MIT.edu
out:            environment variable to assign; Hesiod name of service cluster

Examples...

BIND name:      arktouros.cluster.ns.Athena.MIT.edu
out:            syslib rtsys-e40
out:            usrlib rtusr-e40
out:            printer e40
out:            kerberos e40-kerberos
```

This information will be stored in a set of environment variables refreshed at boot time. This would be accomplished by providing an application "getcluster" which queries the Hesiod database and returns a set of environment variable assignments for the service clusters which could be directly interpreted by a shell. This information is stored in a file */etc/clusterinfo* for later "sourcing" by users' *.login* scripts. If the Hesiod database is unavailable, or no data is returned for the workstation.cluster query, */etc/clusterinfo* is not modified, which implies that the previous successful query to Hesiod is used. At installation time, */etc/clusterinfo* is hard-wired with a set of defaults, which will eventually be overwritten at a later boot time by a successful Hesiod query. "Getcluster" would be invoked in */etc/rc* after the start of the Hesiod name server, but before any queries for file systems. For the use of */etc/rc*, a Bourne shell script, we define a flag, *-b*, to "getcluster", indicating that the program's output should be in Bourne shell format.

Applications which require cluster information, such as the RVD and lpr utilities, would search the user environment for an appropriate environment variable, and retrieve the service cluster name for further resolution.

5.2. File Systems

At present, Athena uses RVD as its sole remote file system, but the scheme we propose should be extensible to other distributed file systems as well, such as NFS, RFS and VICE.

5.2.1. Queries and Data for File Systems.

```
General...

BIND name: filesystem or disk cluster name
out:    filesystem type; server name; filesystem name; default mount point;
        access mode

Examples...

BIND name:      dyer.filsys.ns.Athena.MIT.edu
class/type:     C_HESIOD/T_UNSPECA
```

```
out:            RVD Helen L55 /test1 x

BIND name:      1.00.filsys.ns.Athena.MIT.edu
class/type:     C_HESIOD/T_UNSPECA
out:            NFS Zarquon /usr/courses/1_00 /mnt r

BIND name:      rtsys-e40.filsys.ns.Athena.MIT.edu
class/type:     C_HESIOD/T_UNSPECA
out:            RVD agamemnon rtsys /srvd r
out:            RVD helen rtsys /srvd r
out:            RVD achilles rtsys /srvd r
```

If more than one file system is returned from a query, it is assumed that the name represents a system library, with the semantics that the file systems should be tried in the order given, stopping at the first success.

No spindle information is returned, on the presumption that better ways to handle this RVD-specific feature will be forthcoming.

```
Definitions:  filesystem type: character string from {RVD,NFS,RFS,VICE...}
              server name: machine name in internet domain name space
              filesystem name: name used by server to identify file system
                        for example, could be locker name for RVD,
                        remote mount points for NFS
              default mount point: mount point on local machine
              default access mode: read-only (r), exclusive (x) or shared (s)
                        Note that these semantics are file-system
                        dependent.
```

5.2.2. Impact on *attach*, *detach*, and *rvdflush*. *Up* and *down* have been replaced with analogous programs *attach* and *detach* which query the Hesiod name server. */etc/rvdtab* no longer contains a list of all file systems which the host could ever want to mount. Rather, */etc/rvdtab* contains the list of file systems currently mounted by the host. A successful call to *attach* appends the file system entries to the contents of */etc/rvdtab*. Before performing any mount, *attach* first invokes the VDIOCGETDRIVE ioctl and sends a *flush* command to an RVD server if it is not already providing RVD service to this host.

The syntax of *attach* and *detach* is:

attach *filesystemname* **[-x|-s|-r]**
detach *filesystemname*

By default, *attach* will mount a file system with the mode returned by Hesiod; the *-r*, *-x*, and *-s* flags request read-only, exclusive read-write, and shared read-write access, respectively. *Attach* fails if the server disallows attachment in the requested mode. To attach a private locker, merely give its name:

```
attach dyer-test-locker -x
```

A system library can be attached in the same manner:

```
attach e40-rtsys
```

However, more frequently, the system and user system libraries will be accessed using cluster information previously obtained from Hesiod.

```
# in /etc/rc
# this assigns (at least) the following
# environment variables: usrlib, syslib, kerberos, printer
eval `/etc/getcluster -b`
# sometime later...
attach $usrlib
attach $syslib
```

Also within /etc/rc, before any RVD packs have been mounted, the contents of */etc/rvdtab* will be examined using a *sed* or *awk* script, and an *rvdflush* command will be performed on each RVD server which had been active during the time the system was last up. Following this, /etc/rvdtab will be truncated.

Future versions of *attach* and *detach* which may need to deal with file systems other than RVD packs will make similar queries of the name server.

5.2.3. Source of Authoritative Data. Initially, this data will be hand-entered, being manually transferred from the individuals' locker information maintained by the Athena account administrator as well as from current copies of /etc/rvdtab. Eventually, RVD and other file system information will be maintained by the SMS, and the name server information will be extracted from the SMS and then distributed to the authoritative name servers provided by Hesiod.

5.3. Line Printer Service

The present line printer service within Project Athena is a version of the Berkeley 4.2BSD line printer system adapted for use with the Athena clustertab database. A redesign of the entire printer system based on the BRL MDQS system is presently underway, and we discuss the MDQS system here.

5.3.1. MDQS Printer Information.

```
General...

BIND name:      printername, queuename or service cluster name
out:            printername queuename serverhost ability hardwaretype
out:            printername queuename serverhost ability hardwaretype
                (possibly many resource records returned)

Example...

Hesiod queries:

        hes_resolve(printername, "printer") OR
        hes_resolve(queuename, "printer") => 0 or more resource records
                                             of the form:
        printername queuename serverhost ability hardwaretype

        hes_resolve("woodchuck", "prinfo") =>
        woodchuck woodchuck-text Castor.MIT.edu postscript lps-40
        woodchuck woodchuck-ps Castor.MIT.edu text lps-40

        hes_resolve("woodchuck-text", "printer") =>
        woodchuck woodchuck-text Castor.MIT.edu postscript lps-40
```

Hesiod print clusters may also be resolved with the same HesiodNameType, and resolve to a list of all printers for that print cluster, e.g.,

```
hes_resolve("e40", "printer") => 0 or more resource records
woodchuck woodchuck-ps Castor.MIT.edu postscript lps-40
woodchuck woodchuck-text Castor.MIT.edu text lps-40
homer homer-ps Castor.MIT.edu postscript scout
homer homer-text Castor.MIT.edu text scout
watson watson-text Castor.MIT.edu text 3812
watson watson-dump Castor.MIT.edu 3812wd 3812
```

5.3.2. Impact on Line Printer Software. As distributed by BRL, the MDQS system deals with "print queues", not printers. A queue is a uniquely named object associated with one or more printers. A printer may have more than one queue associated with it, usually to accomodate different ways of using the printer, e.g., text versus graphics dumps, etc. We wish to extend MDQS so that a user can specify the printer, a more general concept than a queue, and add intelligence to the print queuer so that it can itself determine on what particular queue for a printer the file should be placed. In addition, we wish to incorporate the Hesiod name server into the distributed version of MDQS so that configuration information regarding the location and capabilities of printers within Athena can be managed centrally, without the use of replicated configuration files on each workstation.

In our proposed naming scheme for printers, clustertab-derived abbreviations are thrown out. All printer names are global, (i.e., there can be only one printer named 'ln03') although a single printer may have several unique nicknames, or aliases. In the absence of a specified printer, the Hesiod print cluster name made available at boot time is resolved to obtain a list of printers available in that cluster.

The choice of what queue to place the file on (if it is not otherwise specified) is based on the type of data being printed. We assume that all such data is self-identifying (this needs to be verified); logic will be added to the "qpr" spooling program to choose a particular queue based on the first few bytes of a file. The queue types, or 'abilities', are as follows: text, postscript, windowdump (wd), ln03windowdump (ln03wd), 3812windowdump (3812wd) and x9700. These can also be specified on the command line as an argument to the '-A' flag (see below.)

If no printer or queue is specified, the Hesiod print cluster name for the workstation is resolved to obtain a list of all printers in the cluster, and a printer queue is chosen from the list depending on the type of data being printed. If no printer in the cluster satisfies the criterion, an error message will be produced. If more than one printer in the cluster satisfies the criterion, the printer is chosen randomly from the set of matches.

```
qpr [-q queuename] [other options] file1 [file2 ... fileN]

Additional arguments suggested:

        -P printername  - specifies a particular printer (or set of
                          printers.)  This is different from the
                          '-q' switch, in that a printername might
                          specify a number of different queues.

        -A ability      - specifies what "ability" printer you are
                          searching for, e.g., "postscript", "text"
                          "wd", "ln03wd", "3812wd", "x9700"
```

```
-T hardwaretype - specifies a particular hardware type to be
                  chosen in preference to others.  Note that
                  this primarily allows one to choose between
                  different
```

5.4. Toehold Passwd Server

5.4.1. UNIX /etc/passwd information; other user information.

```
General...

BIND name:      athena username
out:            standard /etc/passwd entry

Example...

BIND name:      saltzer.passwd.ns.Athena.MIT.edu
out:
```

```
Saltzer:QwtFRKLKqig:994:64:Jerome H Saltzer,,E40-391AM,3-6016,:
/mit/Saltzer:/bin/csh
```

Password and group information is only useful if a community specified by a domain (as in NS.Athena.MIT.edu) shares the same user-id and group-id space, not to mention the assumptions that all names resolve to UNIX-specific information. This is the current policy within Athena, although it is subject to change.

5.4.2. Impact on Toehold Software. Toehold presently asks the *passwd server* for a password entry for a user. The changes needed to have it query the Hesiod name server are minimal and localized.

5.4.3. Source of Authoritative Data. The database maintained by Athena_Reg is presently the source of */etc/passwd* information provided by the *passwd* server. Already, copies of */etc/passwd* are distributed regularly to the timesharing hosts. A program has been written which converts a */etc/passwd* file to a database file used by Hesiod at boot time. When this file is distributed by cron to the hosts on which the ns.Athena authoritative name server resides, the name server can be signalled to indicate that its initial database should be reread into its virtual memory.

5.5. RHS Hesiod Expansion

This query is used internally within the Hesiod library to expand an unqualified *RHS* of a Hesiod name. Storing RHS mappings within the name server itself allows frequently named objects in other Hesiod domains to be conveniently abbreviated: e.g,,

```
attach mydisk@sipb
```

instead of

```
attach mydisk@sipb.MIT.edu
```

to mount a filesystem located at SIPB.

```
General...

BIND name:      RHS.rhs-extension.ns.Athena.MIT.edu
class/type:     C_HESIOD/T_UNSPECA
```

```
out:            replacement string

Example...

BIND name:      SIPB.rhs-extension.ns.Athena.MIT.edu
class/type:     C_HESIOD/T_UNSPECA
out:            SIPB.MIT.edu

BIND name:      DU.rhs-extension.ns.Athena.MIT.edu
class/type:     C_HESIOD/T_UNSPECA
out:            DU.MIT.edu
```

5.6. Service host location

```
General...

BIND name:      advertised service name or service cluster name
class/type:     C_HESIOD/T_UNSPECA
out:            server hostname

BIND name:      olc.sloc.ns.Athena.MIT.edu
class/type:     C_HESIOD/T_UNSPECA
out:            ringworld.Athena.MIT.edu

BIND name:      e40-kerberos.sloc.ns.Athena.MIT.edu
class/type:     C_HESIOD/T_UNSPECA
out:            Kerberos.Athena.MIT.edu
out:            Kerberos-backup.Athena.MIT.edu

Definitions:    server hostname: hostname in internet domain space.
```

Many resource records may be returned, indicating that several hosts provide this service.

5.7. Proposed Additional Queries

Ideally, the queries and the data formats of their resource records would be decided upon by the designers of the applications which use Hesiod in consultation with the designers of the name server. Because many of the programs which could profitably use the name server have already been written and lack a design team, we have come up with queries which make sense for many applications. These should be taken as working suggestions, not cast in stone, and developer input is welcome.

For each query we describe its format and the contents of the resource records returned.

5.7.1. Post Office Boxes.

```
General...

BIND name:      athenausername.pobox
class/type:     C_HESIOD/T_UNSPECA
out:            postoffice type; servername; postboxname

Example...

BIND name:      lerman.pobox.ns.Athena.MIT.edu
class/type:     C_HESIOD/T_UNSPECA
```

```
out:            POP E40-PO.MIT.edu lerman

BIND name:      saltzer.pobox.ns.Athena.MIT.edu
class/type:     C_HESIOD/T_UNSPECA
out:            repository Heracles.MIT.edu jhsmail2

BIND name:      fhsu.pobox.ns.Athena.MIT.edu
class/type:     C_HESIOD/T_UNSPECA
out:            timesharing Heracles.MIT.edu fhsu
```

Presumably a user agent for reading mail (MH, Gnu Emacs, etc.) could use the 'pobox' query to locate and access the user's mailbox.

5.7.2. */etc/services* replacement.

```
General...

BIND name:      name of service
class/type:     C_HESIOD/T_UNSPECA
out:            protocol decimal port number

Example...

BIND name:      smtp.service.NS.Athena.MIT.edu
class/type:     C_HESIOD/T_UNSPECA
out:            TCP 25
```

6. The Hesiod Library

The Hesiod library is a set of routines which interfaces with the BIND resolver library to provide a means to translate Hesiod names into data which can be used by the applications. In addition to *hes_to_bind*, defined above, the most frequent interface to the library will be through the routine:

```
char **
hes_resolve(HesiodName, HesiodNameType)
char *HesiodName, *HesiodNameType;
```

This routine takes a Hesiod name and its associated name type, translates it (via a call to *hes_to_bind*), and passes the resulting strings to the BIND resolver library. The returned value is a pointer to an array of character pointers or NULL in the case of an error. The array is terminated with a NULL pointer. Each character pointer in the array represents one of the data fields returned in the resource records by the name server.

If hes_resolve or hes_to_bind return NULL, it will be possible to examine a Hesiod-specific error code by invoking the following function:

```
int
hes_error()
```

Hes_error returns 0 where there is no error; otherwise one of the following manifest constants defined in */usr/include/hesiod.h*:

```
#define HES_ER_OK       0 /* no error */
#define HES_ER_NOTFOUND 1 /* Hesiod name not found by server */
#define HES_ER_CONFIG   2 /* local problem; (no config file?) */
#define HES_ER_NET      3 /* network problem */
                          /* can't reach network, no response */
                          /* from servers */
```

7. Deployment

7.1. Deployment of Name Servers

Following the recommendations for the deployment of domain name servers, there should be multiple hosts which are authoritative for the ns.Athena.MIT.edu domain. This will be satisfied by placing a primary Hesiod name server on each of the backup server machines, namely, jason, zeus, apollo, clio and ringworld. The parent domain, presently MIT.edu, which will include the Athena workstations, will been informed of which hosts are the authoritative server.

7.2. Distribution of Authoritative Data

BIND has the concept of both *primary* and *secondary* authoritative servers for a domain. A primary server reads its authoritative data from an ASCII file in a standard format specified by RFC973 when it first starts up. Primary servers can also be made to reread this file on receipt of a signal. A secondary server receives its data from a primary server via a *zone transfer* through a TCP connection, which transfers resource records one at a time between the master and secondary server. A zone transfer occurs at start-up time and periodically during the time the secondary server runs. With large amounts of data in the database, a zone transfer can take a long time. In addition, during the time of the zone transfer, the server is not available to respond to queries. Within Athena, secondary BIND servers have not been widely used due to bugs in the early versions, instead using multiple primary servers. Until more experience has been gained with the use of secondary servers, Hesiod will follow Athena's present method of running independent primary servers. Note that the distinction between primary and secondary servers has been introduced by BIND, and is not relevant to client programs which access the name server.

Each of the primary authoritative servers for the ns.Athena.MIT.edu domain will be distributed a new copy of the database regularly from Hector, or whatever other machine we choose to be responsible for maintaining all authoritative data. Roughly speaking, the method of distribution would be:

```
Extract data from AthenaReg, other authoritative sources
        of information, and place in an ASCII file in
        RFC973 "Standard Resource Record Format."  We
        call this the BIND authoritative database for
        ns.Athena.MIT.edu.
For each authoritative server:
        Copy this file to server.
        Signal name daemon on server that it should
        reread its database.
```

This could be accomplished by running two crontab scripts: one on Hector which would periodically create a new ASCII database file in much the same was that passwd and group files are now created, and on each machine running a primary server, a script would copy the file from Hector and then signal the server to reread the file. The cron scripts on the

primary servers could be synchronized with the distribution of the authoritative data from Hesiod, or they could wake up regularly and check if the modification date has changed.

7.3. Origin of Authoritative Data

As mentioned above, the first utilities which will rely on Hesiod require that the data they use now be translated into a form suitable for input to the name server. There is already a program which formats the full Athena /etc/passwd file extracted from Athena_reg into a file which is RFC973 compatible. The primary copy of */etc/rvdtab* which lists all the system and user RVD libraries and as well as all the RVD private lockers has also been converted to RFC973 format. Remaining to be done are the contents of *clustertab* and *printcap* for use with the Berkeley *lpr* line printer service, as well as the assignment of individual workstations to service clusters. Appendix A is an example of the RFC973-conformant data files which are used by the Hesiod name server.

Until the SMS comes into existence, it will be necessary to edit these files manually, with the exception of the password database, which can be generated from the AthenaReg database on Hector. For the moment, the interface between Hesiod and the SMS will be through the RFC973-format data files described in Appendix A.

7.4. The Problem of Independent Living Groups

We envision that independent living groups will want to run their own authoritative Hesiod name server to manage their own site-specific data. For example, the DU fraternity will be making requests of the RVD and lpr utilities which should necessarily refer to services within their own local network. This could be accomplished by giving each of the ILGs a separate set of cluster resource records, but that would impose the restriction that the data would be owned and controlled by Athena, and that Hesiod queries would cross the slow-speed network link even when services and their clients were both located at the ILG. What's more, if there were ever problems with the link to the main campus, local services would also be unavailable (or at the minimum, hard to reach.)

Thus, we recommend that each ILG run its own authoritative version of Hesiod, initially with hand-crafted data files patterned after the main data files from Athena. For example, to accomplish this, the DU fraternity must customize their own copy of */etc/hesiod.conf*, and designate to their parent server a server to be authoritative for their Hesiod domain.

```
#file /etc/hesiod.conf for the DU fraternity
#comment lines begin with a '#' in column 1
#LHS table
lhs = .ns
#RHS table
rhs = .DU.MIT.edu
```

One question which has arisen is how to refer to frequently-accessed objects outside one's own Hesiod domain. As mentioned earlier, the ability of a Hesiod name to contain an unresolved RHS is a convenient way to accomodate abbbreviations. Thus, if the Hesiod server for DU contains the mapping:

```
hes_to_bind("athena", "rhs-extension") => "Athena.MIT.edu"
```

then it would be quite easy for someone at DU to mount a remote file system (not that it would be pleasant, due to the slow speed of the line):

```
attach 6.001@athena
```

However, if these queries became *very* frequent, it would be possible for the system

administrator at DU to enter CNAME records for locally-named objects which pointed back into Athena. For example, with the example given above, an entry in the Hesiod data file of the form:

```
6.001.filsys    HESIOD  CNAME   6.001.ns.Athena.MIT.edu
```

would allow one to execute the command:

```
attach 6.001
```

This requires close coordination between both sites because fully-qualified names are now being exchanged; however, since ILGs interact closely with Athena staff, this is an attractive shortcut.

7.5. Dynamic Updates to BIND

The BIND system was designed to operate with relatively static data such as is found in the Internet name/address space, meaning that updates would be expected to occur relatively infrequently, once a day or less. This accounts for the all-or-nothing approach to refreshing BINDs internal database which requires rereading the database file from disk or transferring an entire zone. This is feasible when the data in a zone is of moderate size, but quickly becomes cumbersome when large amounts of data are involved.

People at the University of Washington have provided a compile-time switch in BIND 4.5 to allow unauthenticated updates in the form of new "query" opcode types, but full zone transfers will still occur between the primary and any secondary servers. Clearly, if we wished to use this facility as groundwork for a more secure and robust system, we would have to do the following:

Add authentication to the update opcodes.
This should be able to support Kerberos, but the hooks should be general enough that a different site could run Hesiod without requiring Kerberos.

Arrange for changes to be "journaled", so that it would be possible for secondary servers to ask only for the changes in a zone.

Adjust the *time to live* **fields of the data so that caching servers will reflect changes to the master servers relatively rapidly.**

Appendix A. Hesiod RFC973 Data Files

RFC973 specifies the format of a domain server data file, and the complete definition can be found in that report. However, for the purposes of defining the data files used by Hesiod, data items appear in the file in a line-oriented, ASCII format of the form:

```
key     CLASS   TYPE    "data"
```

where *data* represents the content to be returned to a domain server query with the 3-tuple: [key, class, type]. Multiple lines with the same key represent distinct resource records, all of which would be returned in a query. Hesiod uses the class, C_HESIOD, and the type, T_UNSPECA. Thus, an entry for a Hesiod name has the form:

```
HesiodName.HesiodNameType       HESIOD  UNSPECA "data"
```

It is also possible to create aliases for a canonical name using the type *CNAME*. Thus a query for *name1*, *name2* and *name3* would all return the same *data*, given the following lines in the data file:

```
name1   HESIOD  UNSPECA "data"
name2   HESIOD  CNAME   name1
name3   HESIOD  CNAME   name1
```

This provides a convenient way for many names to point to the same data object, and is exploited by the Hesiod *cluster* queries.

The definition of RFC973 allows auxiliary data files to be included using a directive much like C language include files. This makes for easy maintenance and modularity, and is demonstrated in the examples below.

Note that these data files are required to be present on all BIND primary master servers authoritative for a domain. They would not appear on most private workstations or timesharing systems that were not a primary Hesiod name server.

The examples here are *not* authoritative and do not necessarily represent the final data that will be stored by Hesiod for ns.Athena.MIT.edu. Data here is presented for illustration only. The class *C_HESIOD* has been assigned the abbreviation *HS* by Paul Mockapetris.

7.6. Main Database File -- hesiod.db

```
; /etc/athena/nameserver/hesiod.db, Mon Mar  1 14:30:37 EST 1987
;
; Authoritative data for NS.Athena.MIT.edu
;
                IN      SOA     HESIOD.MIT.edu. FHSU (
                                4.7             ; serial - database version number
                                1800            ; refresh - sec servers
                                300             ; retry - for refresh
                                3600000         ; expire - unrefreshed data
                                7200 )          ; min
                 IN     A       18.72.0.46
                 IN     NS      THANATOS.MIT.edu.
                 IN     NS      HESIOD.MIT.edu.
;
; CLUSTER info
;
$INCLUDE /etc/athena/nameserver/cluster.db
;
```

```
; PRINTER aliases
;
$INCLUDE /etc/athena/nameserver/printer.db
;
; FILE SYSTEM
;
$INCLUDE /etc/athena/nameserver/filesys.db
;
; PASSWD info
;
$INCLUDE /etc/athena/nameserver/passwd.db
```

7.7. Cluster Information -- cluster.db

```
e40-rtsys.cluster       HS      UNSPECA "syslib rtsys-e40"
e40-rtsys.cluster       HS      UNSPECA "usrlib rtusr-e40"
e40-rtsys.cluster       HS      UNSPECA "printer e40"
e40-rtsys.cluster       HS      UNSPECA "kerberos e40-kerberos"
e40-vssys.cluster       HS      UNSPECA "syslib vssys-e40"
e40-vssys.cluster       HS      UNSPECA "usrlib vsusr-e40"
e40-vssys.cluster       HS      UNSPECA "printer e40"
e40-vssys.cluster       HS      UNSPECA "kerberos e40-kerberos"
arktouros.cluster       HS      CNAME   e40-rtsys.cluster
goanna.cluster          HS      CNAME   e40-rtsys.cluster
e40-342A-1.cluster      HS      CNAME   e40-rtsys.cluster
bitsy.cluster   HS      CNAME   e40-vssys.cluster
```

7.8. File System Information -- filesys.db

```
rtsys-e40.filsys        HS      UNSPECA "RVD agamemnon rtsys /srvd r"
rtsys-e40.filsys        HS      UNSPECA "RVD helen rtsys /srvd r"
rtsys-e40.filsys        HS      UNSPECA "RVD achilles rtsys /srvd r"
rtusr-e40.filsys        HS      UNSPECA "RVD agamemnon rtusr /urvd r"
rtusr-e40.filsys        HS      UNSPECA "RVD helen rtusr /urvd r"
rtusr-e40.filsys        HS      UNSPECA "RVD achilles rtusr /urvd r"
vsusr-e40.filsys        HS      UNSPECA "RVD andromache vsusr /urvd r"
vssys-e40.filsys        HS      UNSPECA "RVD andromache vssys /srvd r"
vssipb.filsys   HS      UNSPECA "RVD charon vssipb /usr/unsupported/sipb r"
panda.filsys    HS      UNSPECA "RVD priam panda /mit/p/a/panda/Work x"
2.40.filsys     HS      UNSPECA "RVD helen 2.40 /mnt r"
2.40w.filsys    HS      UNSPECA "RVD helen 2.40w /mnt r"
10.13.filsys    HS      UNSPECA "RVD helen 10.13 /mnt r"
10.13w.filsys   HS      UNSPECA "RVD helen 10.13w /mnt r"
dyer-locker.filsys      HS      UNSPECA "RVD helen.MIT.edu L55 /mnt x"
L55.filsys      HS      CNAME   dyer-locker.filsys
```

7.9. Printer Server Information -- printer.db

```
ln03-e40-2.printer      HS      UNSPECA "ln03-bldge40-2 e40-printserver-1"
e40-default.printer     HS      UNSPECA "pp3812-e40-1 e40-315-1"
```

7.10. Passwd Information -- passwd.db

```
ddcote.passwd  HS UNSPECA  "ddcote:ddeDnsY0Av5TY:17358:101:\
David D. Cote,,,,:/mit/d/d/ddcote:/bin/csh"
dyer.passwd    HS  UNSPECA  "dyer:zdoZ1uraJZQF6:17287:64:\
Steve Dyer,Steve,E40-342AM, 0127,4911648:/mit/d/y/dyer:/bin/csh"
```

```
epreid.passwd  HS  UNSPECA  "epreid:epzRhBRgasbac:17400:101:\
Eric P. Reidemeister,,,,:/mit/e/p/epreid:/bin/csh"
```