Network Auto-Magic Design

Version 1.0.5, 2007-Mar-05

John Beck	Jim Carlson	Renee Danson	Michael Hunter
Anay Panvalkar	Kacheong Poon	Garima Tripathi	Jan Xie

There are eight focus areas described below:

There are also two appendices:

---

1. Overview & Component Interaction

1.1 Introduction

Network Profiles, the primary component of the Network Auto-Magic project, are a way to simplify network configuration management. They work by allowing users to specify profiles (i.e. collections of various properties) which determine how things work in different circumstances. The profiles and their respective properties include:

- Network Configuration Profile (NCP)
  - which network interface(s) to use
  - how to obtain IP address(es) for the interface(s) in use
  - whether or not a given link should be configured automatically
  - parallel interfaces to the same subnet (i.e., link aggregation and IP Multipathing)
- Environment
  - conditions under which a profile should be activated
  - which name service(s) to use
  - a host name (and any required variations thereof)
  - routing information
  - a set of IP filter rules
  - `smf(5)` services
  - user-specified post-activation "hook"

This is discussed in much more detail in section 4 below.

1.2 Overview

Let us begin with an architectural overview. The primary components are:

- The profile repository. This is where the configuration program stores its data, which will also be read by the profile daemon.
- The profile configuration program (a.k.a. the UI).
  - Note that there will be both CLI and GUI versions of this program which will perform similar if not identical tasks.
  - In addition to using the repository, it also interacts with the profile daemon.
  - Tasks which users will use this program to perform include:
    - creating, modifying and deleting profiles
    - activating one or more profiles
    - querying information about profiles
- The profile daemon.
  - This reads data from the repository.
  - It reacts to events as notified by the event handler.
  - It reacts to changes which users make via the configuration program.
  - The event handler "state machine" is implemented in this daemon.
  - The daemon also interacts with the SMF network services.
- The event handler, which will gather information about events from the kernel.
- The SMF network services. These are already part of Solaris, but will be modified to some extent. The daemon will restart / refresh some of these services as needed.

1.3 Interactions

How they interact is as follows:

- At any given time, one NCP and one Environment are "active".
- At boot, the profile daemon consults the repository for the current active NCP, proceeds until one or more IP addresses have been configured, checks the conditions of the Environments, activates the one specified by the policy engine, and configures the network(s) accordingly.
- As events occur which may trigger a change in the network configuration, the event handler detects these and the daemon consults the active profiles and may reconfigure the network(s) accordingly. Note that some of these events may indicate that the conditions have changed.
- A change in conditions may trigger a change in the Environment, which may in turn affect the network configuration.
- If a user modifies a profile, the configuration program updates the repository and notifies the daemon. If the current active profile is modified, then the daemon will reconfigure the network(s) accordingly.
- Likewise, if a user activates a new profile (NCP or Environment), then the configuration program updates the repository and notifies the daemon, which will then reconfigure the network(s) accordingly. Note that a user can always manually activate a profile (NCP or Environment), regardless of conditions. Also note that users who desire total control will be able to specify conditions such that a different profile is never activated automatically.

See Section 3 and Section 4 for detailed discussion of how these will all work.

1.4 Examples

- There will be an out-of-the box Environment for "no network" which specifies `files` for everything in `/etc/nsswitch.conf`, disables services which make no sense in a stand-alone environment, etc. Then at boot, the profile daemon would consult the conditions, note that there was no networking, then automatically select that Environment.
- A user at Sun might specify a "SWAN" Environment:
  - conditions of the form "apply when a network with IP addresses in the range `129.144.0.0/12` is detected"
  - a property to use name server X
  - a property to use `files/dns/nis` or `files/nis` in `/etc/nsswitch.conf`
  - a property to use NIS server Y
  - enable the SMF service `nis/client`
  - etc.
- A user might specify an NCP of the form "connect link `ath0` to whatever WLAN, then use DHCP to get an IP address", then a related Environment whose conditions activate it contingent on `ath0` being connected to ESSID and BSSID *Y*, then have a user "hook" to punchin, which creates an IPsec tunnel, thus triggering a condition check, which ultimately leads to the "SWAN" Environment being activated.

2. Information Gathering

Before NWAM can select a profile to activate, it needs to learn about the available network links and the connected networks. This information gathering is done at different levels of the network stack.
For example, at the data link level, NWAM might detect that 802.1X is being used to do authentication and need to consult a link layer profile for authentication information.

Suppose there are two Ethernet interface cards available on a Solaris box and both of them are connected to a network. NWAM might use the Link Aggregation Control Protocol (LACP) to find out if the two NICs can be grouped together to form an aggregation. If it is possible, NWAM will remember this information when selecting a profile.

Another example is when a machine has a WiFi network interface card, NWAM will need to find out all the available WiFi access points before it can decide which one is preferred.

At the IP level, NWAM will also need to find out several pieces of information. Again suppose there are two NICs available on a given box. NWAM will need to find out if the two NICs are connected to the same IP network. If they are, there is a possibility to set up an IPMP group on them to increase network availability. NWAM will also need to find out if IPv6 can be used with the attached IP network. It will use this information later to decide if an IPv6 interface needs to be plumbed over a network link. Another example is that NWAM will need to find out if DHCP service is available on the attached IP network. If it is, NWAM will need to find out what the available DHCP server(s) is/are providing. NWAM can use this information to select which server to talk to or if it should use DHCP to configure an interface at all.

2.1 DHCP

2.1.1 Background

When NWAM detects that an IP interface is available (the IFF_RUNNING flag is set), it needs to decide how to configure the interface by consulting an NCP. An NCP may specify that this interface is to have a static IP address. Or it may specify that this interface is to be configured using DHCP. Or it may specify that depending on different "conditions," this interface is to be configured using different methods. Or it may not specify any method to configure the new interface, and NWAM needs to decide what to do with it.
In the last two cases above, NWAM needs to discover what is available using the new interface. For example, it needs to find out if a DHCP server is available through the new interface. Currently, Solaris' dhcpagent(1M) does not provide a detection method which a caller can request the dhcpagent to find out the DHCP information on a given interface. When DHCP is enabled on an interface, the dhcpagent will try to complete the full DHCP lease processing and bind the interface. This document is to introduce a discover method to the dhcpagent without doing the actual lease processing.

2.1.2 Proposal

1. dhcpagent changes

   In order to satisfy the above need to NWAM, we propose to add a new discover mechanism to dhcpagent. A new DHCP_DISCOVER request is added. When the dhcpagent receives such a discover request, it will send out the DHCPDISCOVER message through the requested interface as if it is trying to obtain a lease of an IP address from a server. After it receives the DHCPOFFER message from a server, it will reply the discover request with an OK. The

agent will remember this DHCPOFFER message for a specific period of time. If dhcpagent receives multiple offers, it will remember all of them. And each offer is assigned a non-zero index. During this period, the internal state of the interface is DISCOVERED. After this period, the state will be changed back to INIT and all the cached DHCPOFFER messages will be freed.

If the dhcpagent receives the discover request on an interface which is already in BOUND state, it will reply the request with an error DHCP_IPC_E_OUTSTATE. As the discover request is asynchronous, the agent may reply with error DHCP_IPC_E_PEND. It is up to the caller to retry.

After the discover request is finished, the requester can then retrieve information from the DHCPOFFER message(s) using the DHCP_GET_ALL_TAG request. This new request can return data either from DHCPOFFER or DHCPACK. The caller can specify which one it wants in the request. If a caller requests data from DHCPOFFER, dhcpagent will return all the received DHCPOFFER messages along with their assigned index to the caller. If a caller requests data from DHCPACK, dhcpagent will return the received DHCPACK message to the caller.

The DHCP_START request is also modified. The caller can specify an index in the request to tell the dhcpagent which DHCPOFFER message to use in generating the DHCPREQUEST message to the DHCP server. If the index is 0, it means that the dhcpagent can use the current rules to pick the appropriate DHCPOFFER message.

If a DHCP_START request comes in for an interface in DISCOVERED state, the dhcpagent will generate a DHCPREQUEST message based on the saved DHCPOFFER message(s). Which message to use is determined by the aforementioned index in the request. If the index is non-zero and the interface is not in DISCOVERED state, the error DHCP_IPC_E_OUTSTATE will be sent back to the caller. After the DHCP lease is acquired, dhcpagent will still cache all the received DHCPOFFER messages for diagnostic purpose. Their information can be retrieved using the DHCP_GET_ALL_TAG request.

2. dhcpinfo(1) changes

   Two new options, -o and -a , will be added to dhcpinfo. If the -o option is set, dhcpinfo will request information from the DHCPOFFER message and print out the result. If not set, it will request information from the DHCPACK message as is done currently. The -o option takes a parameter which is used to specify which DHCPOFFER message to retrieve the result from. If there is no option, it will return results from all the DHCPOFFER messages stored in dhcpagent.

   If the -a option is set, dhcpinfo will print out all fields from either DHCPOFFER or DHCPACK message, depending on whether the -o option is set.

3. ifconfig(1M) changes

   Two new dhcp sub-commands `discover' and `select <x>' are added to ifconfig. The `discover' sub-command asks ifconfig to send the DHCP_DISCOVER request to the dhcpagent. The `select <x>' sub- command asks ifconfig to send the DHCP_START request with an index <x> to the dhcpagent. The current `start' sub-command will send a DHCP_START request with 0 as the index.

2.2 IPMP Groups

Interfaces which are connected to the same Ethernet broadcast domain should be IPMP'd together. Interfaces that are on the same IP subnet will be in the same broadcast domain; thus interfaces that are known to have an IP subnet in common should be placed in an IPMP group together.

The simple subnet test will not catch all interfaces that are connected to the same Ethernet broadcast domain, however. Jim suggests an additional test to discover interfaces that should be in the same IPMP group:

> Instead of pinging (and all the IP and ARP headaches that involves), I'd suggest using a multicast or broadcast message sent to ethertype 9000 (loopback). If the other interfaces can hear your message, then they're in the same Ethernet broadcast domain, and can (and probably should) be IPMP'd together.

2.3 How to configure data-link and IP interfaces

This section discusses the mechanisms NWAM uses to create and configure data-links and IP interfaces.

2.3.1 Overview

In order for a communication pipe to be useful to the user a data-link portion has to be created and configured and an IP interface needs to be created and configured on top of that. It should be noted that a richer structure can be built using tunnels over IP to create another data-link over which IP can be configured and so on.

This section is closely related to Section 3 (Event Handler). That section covers how the mechanisms described in this section are used to manage the lifetime of the data-link and IP interfaces.

Until Clearview has been completely integrated, aggregations, VLANs, and tunnels will not be data-links. This document will discuss the issues surrounding these objects as if they were data-links.

It is expected that the necessary Clearview components will integrate before NWAM is completed. If that does not turn out to be true then an alternate design will be fleshed out and implemented. It is not expected to be difficult.

2.3.2 Creation of data-links

Many data-links are created when the associated driver is attached. We will not discuss those further in this document. For aggregations, VLANs, and tunnels data-links are created using dladm(1M).

The dladm functionality is built upon the consolidation private libraries libdladm (management of link properties), libaadm (configuration of link aggregations), and libiptun (management of tunnels). Configuration of link layer attributes will be done via these libraries.

2.3.2.1 Aggregation

If the link we are bringing up is actually an aggregation of data-links then we need to use the laadm_create() entry point in the laadm library.

2.3.2.2 VLAN

In order to create a VLAN data-link the libdladm function dladm_datalink_create() will be used. Related functions will be used to manipulate the VLAN. See the Clearview "UV Design Specification" and "Link management API design" documents for further information.

2.3.2.3 Tunnel

Tunnel creation will be done with the libiptun function iptun_create(). Related functions will be used to manipulate the tunnel. See the Clearview "IP Tunneling Device Driver Design Specification" for more information. In addition the documents from §2.3.2.2 are relevant.

2.3.3 Configuring IPv4

The following sections will discuss various mechanisms used to apply IPv4 configuration.

When creating and configuring an IP interface at the command level ifconfig(1M) is used.

libinetcfg (PSARC 2001/677, 202/247, 2003/427) was built to provide a rich set of consolidation private routines for configuring interfaces. It is currently not being used by anything within ON. NWAM's needs for configuring IP will be built upon libinetcfg. Due to libinetcfg's lack of use this may cause the need to extend libinetcfg.

2.3.3.1 DHCP

The first mechanism is for the dynamic configuration of IP using DHCP. In order to communicate with dhcpagent the project private library libdhcpagent (PSARC 1999/040) is used. Per §2.1 of this document an additional state is exported from dhcpagent allowing the user to inspect potential leases before accepting one. Minor changes will need to be made to libdhcpagent to generate the new requests.

Depending on the information needs of the NWAM profile being executed the appropriate DHCP command will be sent via the communication channel. If only information is being requested then that will be placed in the appropriate state structures for other components to consume. At a later point the user will either choose to accept one of the leases offered or not continue using DHCP to configure the interface.

2.3.3.2 Static Configuration

If interface configuration parameters are supplied by the user then libinetcfg will be use to configure the interface. Currently libinetcfg does not support the creation of logical interfaces. That functionality will be added to libinetcfg.

Since the removal of logical interfaces might happen in the future, the abstraction used should be consistent with either the use of logical interfaces or the ability to apply multiple addresses to a single link.

Once a logical interfaces (or equivalent abstraction) is obtained then application of the static config is done via icfg_set_{flags, addr, ...}.

2.3.4 Configuring IPv6

The following sections will discuss the mechanisms used to apply IPv6 configuration. A fundamental difference between IPv4 and IPv6 is that the decision to use DHCP is not managed from the host but from the network. Additionally link-local addresses are always available on interfaces and are managed without NWAM's intervention.

2.3.4.1 DHCP

Since stateful address management is controlled from the network the choice to use DHCP is not in the local user's hands. We allow the user to reject DHCP configuration but he cannot request it if the network does not offer it. Unless the user wants to reject DHCP configuration we will allow the dhcpagent to control the interface without our intervention.

The file `/etc/default/dhcpagent` is used to control the agent's actions on each interface.

2.3.4.2 Static Configuration

The library libinetcfg will be used to apply static IPv6 configuration to IP interfaces. Since the addresses are too long for most people to remember, users rarely do this.

2.4 Suspend/Resume

After a suspend/resume or other operation which causes a link flap, we need to re-evaluate our network environment to determine if our configuration is still valid or not. Fortunately, DHCP takes care of this for us, so there is little to do in the general case. There may be some corner cases in which the use of DNA (see RFC 4436 and I-D dna-protocol-03 for details) might provide some degree of optimization, which we may look into in the future if the need arises, but that will be part of a later phase of this project.

3. Event Handler

The NWAM daemon needs to handle many different events triggered both externally by the system or user and internally by its different threads. For example, it needs to handle the event of an IP interface becomes unavailable because the underlying link is gone. It needs to handle internally generated events such as the thread responsible to gather interface information has completed. This section explains the different events related to data-links and how they are handled by the NWAM daemon.

3.1 Events

The following are the explanations of the events the daemon needs to handle with data-link and IP interface.

3.1.1 Link and Interface Events

3.1.1.1 EV_LINK_CREATE / EV_LINK_DESTROY

These events are triggered externally when the daemon detects that a new data-link is created/destroyed. With the Clearview project, a new kind of sysevents will (need to be confirmed) be generated for data-link creation and removal. The daemon will listen to those sysevents and generate the appropriate internal event, EV_LINK_CREATE/EV_LINK_DESTROY. This new kind of sysevents correctly reflects all the link create/ destroy events the daemon is interested in.

The EV_LINK_CREATE event can also be triggered internally when NWAM starts up. At that time, the daemon detects all existing links and may generate this event depending on the start up method. This is explained in the event handling section later.

3.1.1.2 EV_LINK_UP / EV_LINK_DOWN

The EV_LINK_UP event is triggered externally when the daemon detects that a link is now available and the link flag can be marked NWAM_LINK_UP. Similarly, if the link is down, the EV_LINK_DOWN event is triggered and the link flag NWAM_LINK_UP is cleared. The daemon will use the DLPI interface to catch DL_NOTE_LINK_{UP,DOWN} notifications.

For those device drivers which do not send DL_NOTE_LINK_UP, the daemon can try periodically polling the kstat of the underlying device of a physically link to check if its link is up.

For wireless interfaces, this event is generated when the link is connected to a wireless network successfully. This depends on the device driver to correctly send the DL_NOTE_LINK_UP notification to IP.

3.1.1.3 EV_IF_CREATE / EV_IF_DESTROY

These events should be triggered externally when IP interfaces are plumbed/unplumbed. But currently there is no standard way to catch such events. The NWAM daemon only knows about interface plumbing/ unplumbing done by itself. This mechanism needs to be added, possibly by using routing socket.

The EV_IF_CREATE event can also be triggered internally when NWAM starts up. At that time, NWAM detects all existing interfaces and may generate this event depending on the start up method. This is explained in the event handling section later.

3.1.1.4 EV_IF_INFO_KNOWN

This event is generated internally when the daemon finishes gathering network information of an interface. The NWAM daemon may need to find out network information of an interface before it can make the decision on how to configure that interface. For example, it may want to know about the DHCP information before deciding if the interface should be configured using DHCP.

3.1.1.5 EV_IF_UP

This event is generated internally when an IP interface is configured and up. For example, suppose an interface is to be configured using DHCP. After the dhcpagent(1M) sets the interface address and marks the interface up, the daemon will

get the appropriate routing socket messages (RTM_IFINFO and RTM_NEWADDR). Then it will generate the EV_IF_UP event to tell the other parts of the daemon that an interface is up and running.

As indicated above, this event depends on an IP interface to have an address (RTM_NEWADDR) and is marked up (RTM_IFINFO). Both of these conditions must be true for this event to be generated. The daemon will remember if an IP interface has an address set or not and if it is marked up or not.

3.1.1.6 EV_IF_DOWN

This event is generated internally when an interface is unconfigured and down. For example, suppose an interface is configured using DHCP. After the dhcpagent(1M) successfully releases the lease, the daemon will get the appropriate routing socket messages (RTM_IFINFO and RTM_DELADDR). Then it will generate the EV_IF_DOWN event to tell the other parts of the daemon that an interface is down.

This event will be generated when either an IP interface is marked down or the address is deleted.

3.1.2 Event handling

When starting up, the NWAM daemon reads in the Network Configuration Profile (NCP) as described in Section 4.1 "Network Configuration Profile Contents"). It then creates an internal representation of the NCP and all the Network Configuration Units (NCU). All the link structures are marked with the NWAM_LINK_NOT_PRESENT flag. After the event handler is up, the NWAM daemon will detect the existing network configuration of the system. Then it will perform either one of the following.

1. If the start up method is to clean up everything, the daemon will unplumb all detected interfaces and remove all virtual links. Then for each detected physical link, the daemon will follow the contingency rules as specified in the NCP and NCU(s) to decide if it needs to generate the EV_LINK_CREATE event to trigger the creation of the associated links and interfaces as described in the NCP.

2. If the start up method is not to change the existing set up, then for each detected links and interfaces, the daemon will compare it with the internal representation of NCP and note the difference. If a link and an interface does not exist in the NCP, the daemon will trigger a EV_LINK_CREATE or EV_IF_CREATE event.

3.1.2.1 EV_LINK_CREATE

The daemon will create an internal structure for this link if it is not already created when the daemon starts up. The newly structure does not have the NWAM_LINK_NOT_PRESENT flag set. Based on the information it gets with this event, the daemon will try to find a correct position in the graph of links to place this new link. For example, if this link is a physical link, the daemon will place it as a root link. But if this link is a VLAN link, the daemon will place it as a child of the link it is based on. The link flag of the newly created representation is marked NWAM_LINK_EXTERNAL. The daemon will also check if the new link is up. If it is up, the daemon will trigger a EV_LINK_UP event.

If the structure of this link is already there, the daemon will clear the NWAM_LINK_NOT_PRESENT flag and check if the link flag NWAM_LINK_ENABLED is set. If yes, the daemon will check if the link is UP and if yes, it will trigger a EV_LINK_UP event. If the NWAM_LINK_ENABLED flag is not set and the daemon gets this event, it means that an external entity has created this link. The daemon will stop the event processing.

The daemon will also create a new SMF service instance of the link as explained in Section 4.1.3.

3.1.2.2 EV_LINK_DESTROY

The daemon will first start the interface NCU handling, as explained later, of this event for all the associated interface NCUs of the link. And then it will mark the link flag NWAM_LINK_NOT_PRESENT. It will walk through all its children link and mark their flag NWAM_LINK_NOT_PRESENT. The exception is for an aggregation link which still has at least one link belonging to that aggregation without being marked NWAM_LINK_NOT_PRESENT. The daemon will also destroy the corresponding SMF service instance of the link, as explained in §4.1.3.

3.1.2.3 EV_LINK_UP

The daemon will check the link flag of the corresponding link structure. If it is already set, the daemon stops the event processing.

If the flag is not set, the daemon will set the flag. Then it will walk through all the children of the link and trigger a EV_LINK_UP event for each of them if it is marked NWAM_LINK_ENABLED. If there is no child link, then the daemon will start the interface NCU handling as described below for all the associated interface NCUs. The daemon will also create a new SMF service instance of all the associated interface NCUs as explained in Section 4.1.3.

3.1.2.4 EV_LINK_DOWN

In getting the EV_LINK_DOWN event, the daemon will clear the NWAM_LINK_UP flag of the corresponding link. It will then walk through all the children of that link clearing the NWAM_LINK_UP flag. If a child link is an aggregation and there is another link belonging to that aggregation which still has the NWAM_LINK_UP flag set, the aggregation link flag is not touched and the traversal of link will be stopped for that particular branch. For all the children links which have the NWAM_LINK_UP flag cleared but have the NWAM_LINK_ENABLED flag set, the daemon will start the interface NCU handling of this event as described below in "IP interface NCU handling" for all the associated interface NCU(s).

3.1.3 IP interface NCU handling

Each interface NCU is represented by a structure and the daemon runs a state machine for it. The following are the possible states.

1. NWAN_IF_INIT

2. NWAM_IF_QUERYING

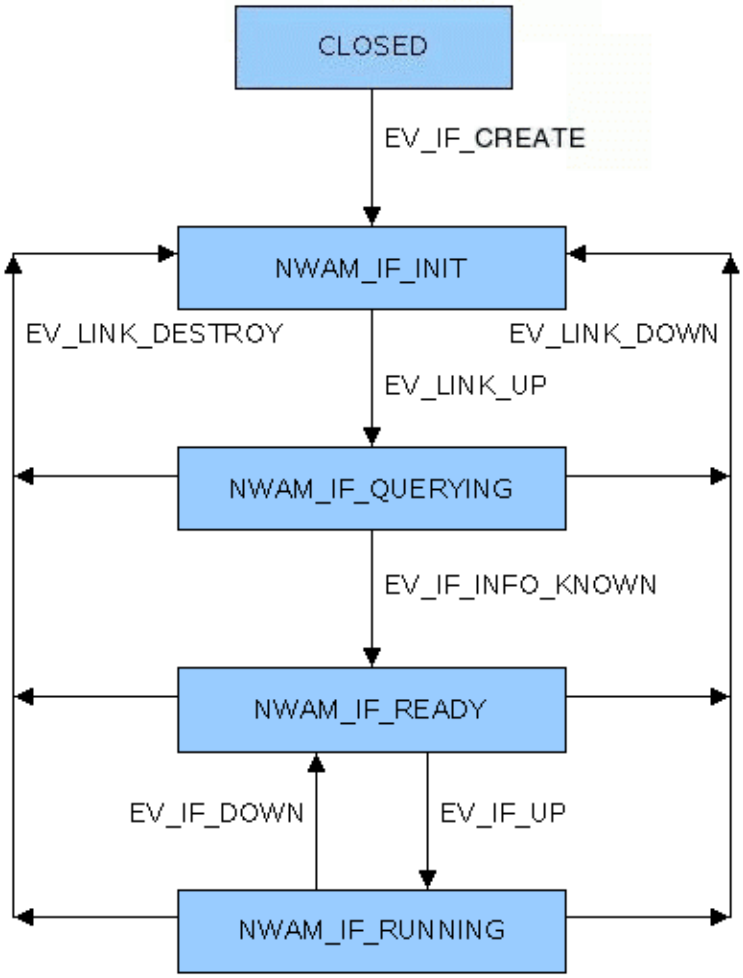
3. NWAM_IF_READY

4. NWAM_IF_RUNNING

For every structure, the initial state is NWAM_IF_INIT. For each interface NCU, the daemon creates it when it reads in the NCP. For externally created interface, the daemon triggers the EV_IF_CREATE event and a structure will be created.

This is a brief state transition diagram of the interface NCU structure. Note that the EV_IF_DESTROY transition is not shown. The reason is that the EV_IF_DESTROY transition is also dependent on the EV_IF_EXTERNAL flag. This flag is set for externally created interfaces not in the NCP. Instead of adding a new set of states just for those interfaces, the flag is used to differentiate them. So the states are the same for both types of interface. An EV_IF_DESTROY event will remove the structure for externally created interface (meaning going to CLOSED state). But for an interface in a NCP, the structure stays and the state will transition to NWAM_IF_INIT.

The interface state transition diagram is as follows;



3.1.3.1 CLOSED

EV_IF_CREATE: the daemon will create the structure representing this interface. The flag of this newly created interface is set to NWAM_IF_EXTERNAL. The state of this interface structure is set to NWAM_IF_RUNNING. An the daemon will find out information about this interface and use the gathered information to fill in the structure.

3.1.3.2 NWAM_IF_INIT

EV_LINK_DESTROY: the daemon will unplumb the IP interface.

EV_LINK_UP: if the flag does not have NWAM_IF_EXTERNAL set, the IP interface is plumbed if it is not plumbe yet and the state will be changed to NWAM_IF_QUERYING. The state machine will start a thread to collect information on the network.

EV_IF_DESTROY: if the flag has NWAM_IF_EXTERNAL set, the daemon will free up the interface structure. The daemon will also destroy the corresponding SMF service instance of the interface, as explained in §4.1.3.

All other events: do nothing.

3.1.3.3 NWAM_IF_QUERYING

EV_IF_INFO_KNOWN: the state will be changed to NWAM_IF_READY. The state machine will use the information stored in the interface NCU to determine how to configure the IP interface.

If only static IP addresses are needed to be set, the IP is configured and the state will be changed to NWAM_IF_RUNNING immediately.

If further processing are needed, such as finishing up the DHCP processing, the state machine will start a thread to do the job.

EV_LINK_DESTROY: the state machine will unplumb the IP interface and change the state to NWAM_IF_INIT.

EV_LINK_DOWN: the state will be changed to NWAM_IF_INIT. Note that the interface is not unplumbed.

EV_IF_DESTROY: the state will be changed to NWAM_IF_INIT.

All other events: do nothing

3.1.3.4 NWAM_IF_READY

EV_IF_UP: the state will be changed to NWAM_IF_RUNNING.

EV_LINK_DESTROY: if a thread has been spawned to configure the interface, the state machine will make a note and the thread will de-configure the interface. If there is no configuration in progress, the state machine will unplumb the IP interface. The state will be changed to NWAM_IF_INIT.

EV_LINK_DOWN: if configuration is in progress, the state machine will make a note and the thread will not configure the interface. The state of the link will be changed to NWAM_IF_INIT. Note that the interface is not unplumbed.

EV_IF_DESTROY: the state will be changed to NWAM_IF_INIT.

All other events: do nothing

3.1.3.5 NWAM_IF_RUNNING

EV_IF_DOWN: if NWAM_IF_EXTERNAL flag is not set, the state will be changed to NWAM_IF_READY.

EV_LINK_DOWN: if NWAM_IF_EXTERNAL flag is not set, the daemon will de-configure the IP interface. The state will be changed to NWAM_IF_INIT.

EV_LINK_DESTROY: if the flag NWAM_IF_EXTERNAL is set, the daemon free up the structure. Otherwise, the daemon will unplumb the IP interface and the state is changed to NWAM_IF_INIT.

EV_IF_DESTROY: if the flag has NWAM_IF_EXTERNAL set, the daemon will free up the interface structure. The

daemon will also destroy the corresponding SMF service instance of the interface, as explained in Section 4.1.3. If the flag is not set, the state will be changed to NWAM_IF_INIT.

All other events: do nothing

4. Profiles

4.1 Network Configuration Profile Contents

See Architecture §4.1 for background and §5.1.1, item 1 for a related discussion of how this will be stored.

4.1.1 Previously Considered Alternatives

Originally we had N separate Link-Layer Profiles (LLPs), one per link, but these were perceived to be too complex to be workable when putting the whole system together. So we came up with the idea of a single Link & Interface Specification (link-spec), but this seemed to be too inflexible, and people had a hard time understanding it.

4.1.2 Proposal

To solve these problems, we returned to the idea of LLPs, using them as a starting point. We then added some attributes to make it easier to build these units into a complete picture. The result is something akin to LLPs which we have dubbed NCUs (network configuration units), which when all put together comprise something akin to a link-spec: an NCP (network configuration profile).

An NCP may also contain information about external network applications that may be registered with NWAM, allowing them to be enabled/disabled by NWAM according to rules defined by the user. This registration interface is discussed in §4.5.2.

4.1.3 Details

There are several attributes which make up an NCU, each of which can take one of several values, some of which have sub-values. In the internal representation of the configuration, we will have a subset of this structure per address. Thus the attributes in this structure possible describe a set of NCUs:

- What is this NCU named? (Note that the name is derived by the system from the underlying link name.)
- What kind of NCU is this?
  - link
    - must specify link class:
      - phys
      - iptun
      - aggr
      - vlan
    - acceptable address acquisition methods (for interfaces which may be above this)

        - static IPv4
        - DHCPv4
            - how long to wait for DHCP server?
            - (optional) static address to use as fallback
        - stateless address auto-conf (i.e., router)
        - stateful address auto-conf (i.e., DHCPv6)
        - static IPv6
    - IP address
        - must specify interface class:
            - simple
            - ipmp
            - loopback
            - vni
        - must also specify IP version:
            - 4, must also specify source of IP address:
                - DHCP
                - static
            - 6, must also specify IP address source:
                - accept stateless address auto-conf (i.e., router)? [Y]
                - accept stateful address auto-conf (i.e., DHCPv6)? [Y]
                - static address & prefix? (may be multiple) [none]
- When to activate?
    - whenever possible [this is the default for all links]
    - contingent on another NCU; a typical example is ath0 being brought up contingent on bge0 not being up
    - don't touch this: NWAM will leave links and IP interfaces of this type (henceforth referred to as "manual NCUs") alone

  Note 1: changes in network configuration are expected to be made using the NWAM UI, to take effect on the next service or instance refresh or system reboot, which would be triggered as part of a "commit" or "apply" operation in the UI. Regarding changes made directly to data links or IP interfaces (other than manual NCUs) using some other UI:
    - Persistent changes (e.g., those made using `dladm(1m)`) will be applied to the current NCP and thus applied to the running system on the next service or instance refresh or system reboot.
    - Transient changes (e.g., those made using "dladm -t" or `ifconfig(1m)`) will not be undone proactively, but they will not persist beyond the next service or instance refresh or system reboot.

  Note 2: to facilitate the management of all this, NWAM will create service instances dynamically, one per NCU. This will allow finer granularity, as instances can be refreshed individually rather than having to refresh the entire service. Because of the dynamic nature of this instance creation, it is inappropriate for any higher level services (whether system provided or user added) to create dependencies on any of these instances.

  Note 3: other projects have proposed similar "instance per link" ideas in the past, largely for fault management purposes. In this case, the motivation is for configuration management, though the result may be serendipitously useful for fault management as well.

Note 4: the proposed names for these dynamic instances are:

- svc:/network/<TBD NWAM service name>:link-<NCU name>
- svc:/network/<TBD NWAM service name>:ip-<NCU name>

though we may end up adding more .X classifiers to the instance name to specify the link/interface class, IP version, etc.

Note 5: during design, a request was made for the ability "to set a static IP for one environment and obtain a DHCP address in another, and have NWAM select appropriately." This use case was deemed sufficiently narrow (DHCP is available just about everywhere) that it would not justify the technical lengths required to support such a feature. The optional static address to use as a fallback in the DHCPv4 entry above was deemed an appropriate compromise.

- What kind of NCU(s) is/are below this? Depending on whether this NCU is a link or an IP interface, and which class thereof, only certain values will be allowed.
  - nothing
  - some link(s)
    - specify link class(es) (if multiple, must all be the same)
- What kind of NCU(s) is/are above this? Depending on whether this NCU is a link or an IP interface, and which class thereof, only certain values will be allowed. Note that we have both 1:N and N:1 relationships between aggregations/IPMP and VLANs. With aggregations all the link types below have to be the same. With IPMP it is more complex: all links in an IPMP group have to be the same link type (e.g., all DL_ETHER). Further, a single IPMP group cannot exist across multiple VLANs (by definition, all IP interfaces in a group must be in the same link-layer broadcast domain). So, IPMP and VLANs only interact one way: IP interfaces on the same VLAN must be placed into the same IPMP group.
  - nothing
  - simple interface
  - ipmp interface
  - aggr link
  - vlan link

Taken together, the entire set of NCUs will comprise the system NCP. Except in the stand-alone scenario, there will be multiple (at least three: loopback + 1 link + 1 IP interface) NCUs in an NCP. The system will only have a single NCP active at any time; although cloning and switching between NCPs will be supported, neither of these is expected to be necessary for "simple" configurations like laptops or desktops.

4.2 Multiple Active Links

The primary design goal here is that, whenever possible, things should Just Work™, with minimal manual intervention. Since we have the technology to plumb all interfaces at boot, and bring them all up (`` `ifconfig *IF* dhcp` `` as needed, possibly preceded by `` `wificonfig -i *IF* autoconf` `` or equivalent), the key question is when we would want to do that, or more to the point, when we would **not** want to do that. In other words, we can already implement the mechanisms, but we need to be able to implement various policies.

So we will need the UI to allow users to specify policies, but we also need to have a default policy (i.e., for when the

user has failed to express his/her preferences for a given scenario). The proposal is that by default all interfaces are plumbed and brought up, though wireless networks will only be connected to after explicit authorization from the user.

If multiple IP interfaces get brought up, then IPMP will be used as possible. IPMP does not work with DHCP, though this is being fixed by Clearview's IPMP Rearchitecture project. So until this is fixed, when multiple IP interfaces are detected to be in the same broadcast domain, only one will be brought up, and it will be brought up using DHCP but not IPMP. Once the IPMP Rearchitecture project is complete, all such interfaces will be brought up using DHCP and IPMP.

This should allow us to do the Right Thing™ in the horizontal scaling and large server scenarios, yet still be flexible for the laptop scenario. The rationale for this set of defaults is Requirement #17:

> For security reasons, bridging networks must be not permitted by default.

(Note that we tend to think of this internally as "thou shalt not have a SWAN interface and a non-SWAN interface up at the same time".) It is fairly safe to assume that if a cable is plugged into an Ethernet port of a machine, then the user wants the machine to bring up **the** network to which the cable is attached, whereas a wireless interface could pick up an AP from just about anywhere, and choosing from among **multiple** networks (and even worse, multiple unknown networks) without data is a potential security hazard.

4.3 Environment Content

See Architecture §4.2 for background and §5.1.1 for a related discussion of how this will be stored.

An Environment (previously called an Upper Layer Profile or ULP) contains information to configure the system after IP service is available. As opposed to NCP attributes which are per-link or per-interface, each of these Environment attributes affect the entire system.

1. Name-services information: what name service to use. More details in §4.3.1
2. Identity:
   - Node name
   - NFSv4 domain
3. proxy servers
4. configuration files for various services, including, but not limited to:
   - IP Filter
   - IPsec
5. SMF services to enable or disable (see §5.4 for a complete list of services)

4.3.1 Duckwater

The Duckwater (aka Simplified Name Service Management) project will introduce name service profiles, which are collections of name service configuration data. These collections will include both name service switch (which name service back-end to use for different types of lookups) information and configuration details for specific back-ends (e.g. DNS, NIS, LDAP).

The general rule we are following for NWAM is that the NWAM Environment specifies the enabled/disabled state of

certain upper-layer network services, but the detailed configuration and policy rules for those services belong in the services themselves.

Thus for name service configuration, NWAM will allow the user to specify a name service profile as part of each Environment: when setting up an NWAM Environment, "Name Services" will be a tab, menu item, whatever (depending on GUI details). When you choose this item, you will be presented with a single list of options:

> Automatic
>
> list of existing name services profiles to choose from
>
> "specify name service configuration" which links to Duckwater config GUI

The Automatic option would be the default and would map to a Duckwater profile with all properties set to "automatic." These are described in the Duckwater design draft

> auto value for nsswitch/* properties means that the name service switch (nscd(1M)) should decide which source types to use based on the back-end availability information provided to it by back- ends and delivered using nsutil(1M) [nsutil].
>
> auto value for nsbec/* property means that special NSBEC instance for particular source type is used which would have all appropriate property values set to auto meaning that the configuration for given type should be auto-configured (name service servers discovered using DNS or DHCP etc.).

[Note that NSBEC stands for Name Service Back End Configuration.]

The link to the Duckwater/name service configuration GUI could be a menu item, or it could be a button in the Name Services window.

4.3.1.1 Alternatives Considered

- A somewhat-less-minimal way of doing this would be to allow the user to choose a particular name service, and then click into Duckwater to configure that particular back-end. There are a few problems with this option. First, it does not play well if the user wants to use multiple back-ends--SWAN's DNS/NIS blend, for example. So it may not be an either-or choice. Second, the Duckwater architecture is being designed to be extensible, to allow addition of new back-ends. Building back-end knowledge into NWAM would be an issue if a new back-end were added.
- A simple DNS config could be specified in the NWAM GUI, while more complex configurations are punted to tools introduced by Duckwater. While this does seem convenient for the simple case, it muddles the distinction between the name service configuration subsystem and the NWAM subsystem. The ideal scenario is to keep the two architecturally separate but still have a cohesive user experience, and that should be doable.

4.3.1.2 Concerns

- The Duckwater team is not including a GUI in their phase 1 plans. One plus here is that Duckwater profiles are an instance of Enhanced SMF profiles; so they will tie in to the Visual Panels work, which may help with the creation of a name service configuration GUI. However it happens, having a name service configuration GUI is a requirement for NWAM. If that GUI needs to be implemented as part of the NWAM project, the boundary between it and the

NWAM GUI should be clear so that future Duckwater enhancements could be made cleanly without changes to the NWAM GUI. Creating a separate panel for name service profile configuration would likely be the best approach here.

4.3.2 Enhanced SMF Profile Framework

NWAM profiles will be instances of Enhanced SMF profiles. The Enhanced SMF Profile model allows SMF service properties to be grouped into profiles, which may then be layered on the running system. In this model, NWAM profiles will be layered above the base and the system default (`generic_open`, for example) profiles. The `local` profile will include changes introduced by the user, and will overlay any NWAM profiles.

4.3.2.1 Secure By Default and Enhanced SMF Profiles

The Enhanced SMF Profile team is planning to introduce new system profiles to reflect Secure By Default requirements. The current proposal is to break `generic_open` and `generic_limited_net` into four new profiles:

- `system_services`: services for the system, no public interfaces
- `local_services`: services for local users, like dtlogin, rpcbind in local-only mode, sendmail in local-only mode
- `secure_net_services`: sshd
- `network_services`: services offered to the network, including rpcbind and sendmail in normal mode

The `system_services` and `local_services` profiles will be applied by default, layered below any NWAM profiles. The `secure_net_services` and `network_services` profiles may be used as templates for the creation of new NWAM profiles.

4.3.3 host names

Currently if the user wants to change the hostname of the system, s/he has to either sys-unconfig then configure the system, or update a bunch of files then reboot the machine.
NWAM can introduce an interface for setting the hostname and subscribing to hostname changes. To achieve this, the following steps need to be taken:

1. Use system/identity:node to keep track of system's hostname.
2. When the user changes the hostname through the NWAM UI, NWAM will need to update the node name in the kernel, then update /etc/nodename and /etc/inet/hosts.
3. Refresh system/identity:node
4. All services that use hostname as a rendezvous point, such as autofs and keyserv, they should either be reworked to use "localhost" rather than the current hostname. Or they should build a "restart_on=refresh" dependency with system/identity:node.

4.4 Environment Policy

This section describes the way a user can create Environments for NWAM and also how the policy for selecting the appropriate Environment will work.

4.4.1 Creating an Environment

The creation of a new Environment would be explicitly initiated by the user via the User Interface (UI). When the user indicates that s/he wishes to create a new Environment, the UI would enable the user to provide the list of attributes associated with that Environment. This set of attributes (or outputs) would be available when the Environment is active. See §4.3 just above for a list of these attributes.

Due to the system-wide nature of these attributes, only one Environment may be active at any time. Based on each attribute selected, a detailed description of services/rules and/or configurations applicable to that particular attribute will be presented. So, for example, if the user wants to specify a particular node name for the machine, and so selects the attribute "Node Name", the UI will prompt him/her for the node name s/he wants to give the machine.

The user will also need to specify some sort of selection criteria (or input) such that the NWAM daemon can find the best-fit Environment, given the information available. These selection criteria include IP address, domain name, ESSID and/or BSSID to connect to (in case of wireless) and possibly the active link available.

4.4.2 Selecting an Environment

By default, NWAM will provide two Environments - a "No Network" Environment and an "Automatic" Environment At the time of selection of Environments, the NWAM daemon will first check for the number of active links available. If there are no active links available, then the daemon will not try to apply any Environment, neither user-defined nor "Automatic". Instead, it will assume that the "No Network" Environment should apply, and that the attributes for the machine in the stand-alone mode are required. However, if an active link is discovered, the daemon will examine the Conditions of each Environment and select the one which matches. If none match, the "Automatic" profile will be available as a fall-back, allowing the NWAM daemon to connect to some available network (per Requirement #7). If more than one Environment has Conditions which are met, the Priority of the matching Environments will be used as a tie-breaker.

4.5 External Network Modifiers

The NWAM solution must be able to coexist with external entities, in particular those provided by third parties, which may perform their own link/IP configuration. We will henceforth use the term External Network Modifiers (ENM). Configuration performed by an ENM cannot be stored within the NWAM framework; this would require that the user understand the details of how the service is configured, or that NWAM be designed with that knowledge, for all possible services. Neither of these alternatives is practical.

§4.5.1 describes the coexistence of NWAM and ENMs when they have no knowledge of each other. §s 4.5.2 and 4.5.3 discuss an interface provided by NWAM which will allow better integration of NWAM and ENMs.

4.5.1 Externally-Initiated Configuration

The model described in §4.1 allows externally-initiated configuration; if an ENM (such as a VPN application) does any

new plumbing, NWAM's view of the system will be updated to reflect the changes. This will be done by creating in-memory representations of these new links and/or IP interfaces; as such, these changes are considered transient and not part of the NCP stored in the repository.

When the NWAM service is restarted or refreshed, transient changes will not be recreated. It is not feasible for NWAM to attempt to recreate a given configuration for which it does not have a complete description; nor is it reasonable to extract the NWAM-controlled part of the configuration and only reset that part. Rather, the restart/refresh request must be treated as a request from the administrator for NWAM to restore a specific configuration, i.e. the current NCP stored in the repository.

If the ENM is an SMF service, it can be restarted automatically in this case with a dependency (with `restart_on refresh` attribute) on the NWAM service, but non-SMF services will need manual intervention to recover.

4.5.2 Service Registry Interface

ENMs may be registered with NWAM, allowing NWAM to correctly enable and disable them as needed. The registration may be performed by the ENM itself via an NWAM-provided API, or by the user via the NWAM UI.

An "unregister" operation will also be provided, removing a registered ENM.

Registered ENMs will be instances of the NWAM service. Attributes will include:

- name
- when to activate: whenever possible, contingent (on an NCU), or on user request
- start and stop property groups
- Environment: the ENM may identify an Environment which should be activated when it is enabled.
- state: enabled/disabled/maintenance; set by NWAM. Depends on the last action taken (start/stop, and whether or not it completed successfully).

When an ENM is registered with NWAM, its instance will be created; it will not be tied to the current NCP. Thus, if the NCP is changed, the ENM registration will not be affected.

4.5.3 API for External Network Modifiers

In addition to the service registry, NWAM will offer several control hooks to ENMs. ENMs need not be registered with NWAM to take advantage of these control hooks.

4.5.3.1 Environment Selection Engine

ENMs may request that the Environment selection engine be run; this may be desired if the ENM makes changes which may not automatically trigger the engine (such as bump-in-stack VPNs, which do not create any new links or system changes detectable by NWAM).

An additional hook will allow ENMs to request that the selection engine **not** run until a further request to run it is received. This may be desirable if the ENM is making a series of configuration changes, and it wants to be sure that NWAM does not react until all of its changes have completed. A timeout must be included with this request.

4.5.3.2 Environment Specifications

NWAM will also provide an interface for external entities, including ENMs, to load complete Environment specifications, including conditions under which the Environment should be applied. Thus, an NWAM-aware ENM would be able to specify the Environment that should be activated when the ENM is enabled.
These Environment specifications will be visible through the NWAM UI, and may be modified by the user, as with any other Environment specification. The intention here is to allow an application to install itself such that minimal user intervention is required; but a user who wishes to further customize may certainly do so.

4.6 Wireless

4.6.1 Prioritizing of WLANs

The NWAM daemon maintains a "Preferred WLANs List" which contains the information of all the WLANs that the system has connected to and the used has asked NWAM to remember. This information includes the ESSID, BSSID, the priority of each WLAN, and the type of encryption/authentication used (if any). When a system needs to connect to a network, the daemon scans for the available WLANs. This list of available WLANs is henceforth referred to as the "available WLAN List".
Since it is quite possible to have multiple known WLANs available at the time the system needs to connect to the network, it is required that some kind of prioritization be applied to the "Preferred WLANs List". NWAM will take a simple approach to the prioritization - it will allow the user to specify the order in which the known WLANs should be tried in order to connect, and, in the event that the user has not explicitly specified any order, it will assume that the known WLANs should be tried in the order in which they appear on the "Preferred WLANs List". The NWAM daemon will use the following policies to create and maintain the "Preferred WLANs List" -

- When the user selects an available WLAN to connect to, he has the option to "remember this WLAN". If this box is checked, the WLAN will be appended to the known WLAN list.
- The user may explicitly add to (or remove from, or reorder) the known WLAN list via the UI's "Configure" option.
- The UI will allow the user to enter WEP/WPA-PSK "keys" when connecting to networks secured by WEP/WPA and have the choice of whether or not to save these "with" the "Preferred WLANs List".

The "Preferred WLANs List" will be used in the following ways by the nwam daemon to decide which WLAN to connect to:

- If the "Preferred WLANs List" is empty, then the user will be presented with the "available WLAN List", from which he can pick the WLAN to connect to. This list will be ordered in terms of signal strength - the WLAN with the highest signal strength will be at the top. The user can pick whichever WLAN works best for him.
- If the "Preferred WLANs List" is not empty, but none of the available WLANs matches the WLAN entries in the "Preferred WLANs List", then the user is presented with the "available WLAN List" (ordered in terms of the signal strength), and is asked to select a WLAN to connect to.
- If the "Preferred WLANs List" is not empty, and only one of the WLAN entries on the "Preferred WLANs List" matches the available WLANs, then the NWAM daemon automatically connects to this WLAN.
- If the "Preferred WLANs List" is not empty, and more then one WLAN entries in it match the available WLANs, then the order of the "known WLAN List" will be followed. Thus, the daemon will try to connect to the entry

appearing closest to the top of the "Preferred WLANs List". In the event that this connection cannot be established, the next highest entry in the "Preferred WLANs List", which also appears on the "available WLAN List" will be tried, and so on. The connection to the highest priority known WLAN would be established irrespective of the signal strength it displays. This might seem a bit arbitrary especially if the known WLAN signal strength is low and the signal strength of a new or lower priority WLAN is high, but since the user has already indicated a preference, it will be followed. The UI will provide a facility to the user which will display the signal strengths of all the WLANs available. If the user gets too frustrated by the connection quality, he can check for the signal strengths of other available WLANs, and select another WLAN with a stronger signal to connect to.

- If the system is already connected to a known WLAN and another WLAN becomes available which is a new or lower priority known WLAN but with higher signal strength, the daemon will not undo an existing established connection. If the user gets too frustrated with the connection quality, he can use the UI facility to scan for all available WLANs and can change the WLAN to connect to.
- If the system is connected to an WLAN which is not in the "known WLAN List" (or is maybe of a low priority), and a new WLAN becomes available which has a higher priority in the "Preferred WLANs List", the daemon would indicate this development to the user. This would most likely be done via visual flag on the Panel.

It should be noted that any time a switch takes place between WLANs, it can affect some currently running applications, and therefore it is better to warn the user that doing so would affect currently running TCP applications. Currently neither Windows nor Mac OS prompt any such warning.

Non-broadcast WLANs are those that don't show up in the `wificonfig scan`. They require that user inputs the ESSID at the very first time. The NWAM daemon keeps track of all non-broadcast wireless LANs which a user has specified, as described in the discussion above about the Preferred WLANs List. This would include both the non-broadcast WLANs that it has successfully connected to, and also those that the user has provided but may not have been connected to yet, unless the user specifically deletes the entry from the "Preferred WLANs List".

Only one "Preferred WLANs List" is maintained for the system instead of a per NCU list, which will include both the broadcast as well as the non- broadcast WLANs. The biggest reason for maintaining a single list is to avoid the deadlock situation at boot time. The NWAM daemon has to know which Profile would need to get activated when the system boots up. However, to decide which Profile gets activated, the daemon would have to know which WLANs are available, which makes it a deadlock if the "Preferred WLANs List" is kept per Profile. The "Preferred WLANs List" will be stored in the SMF repository (libscf.3LIB). The "available WLAN List", however, will be created per NIC, but all the respective "available WLAN Lists" for all NICs will be rolled into one master "available WLAN List" to be presented to the user.

As mentioned above, the user decides the order in which WLANs appear in the "Preferred WLANs List". Whichever WLAN has been assigned the highest ordering by the user, will be tried first (irrespective of whether it is a broadcast WLAN or a non-broadcast WLAN).

Note: during design, it was suggested that "many users would like to choose between available known networks at connect time rather than having to make a persistent decision which will often be wrong". As we did not have sufficient data about which model would be better, we chose the one which we thought was more "auto-magic".

4.6.2 Periodic Scan of Wireless Access Points

The daemon would have to perform periodic scan of all available wireless APs to ensure completeness of information about the network. The scan needs to be done periodically so as to make sure that any WLANs that were not visible at boot time (or connection establishment time), but have become available later, do not get ignored. This is especially important in situations where the WLAN which has become available later is actually present in the "known WLAN List" with a higher priority than the one with which connection is currently established.
As of now, the scan will be done on a fixed interval, the exact duration of which will be determined later. Apart from this, scan should also happen every time the user uses the UI option to list the available WLANs.

Effect of frequent scans on the system:

1. CPU impact - Based on a high level overview of the existing code (usr/src/uts/common/io/ath/) it seems like the probe requests sent out are just like any other network packets sent out, and the response is also treated like normal networking packets received. It is, therefore, safe to assume that they carry minimum impact. They generate an interrupt, but do not over-tax the CPU.
2. Battery impact - This is a bigger cause of concern since frequent scanning would drain battery power. Also, this impact would differ on different types of systems based on their make and manufacturers. It is not clear how exactly to address this issue.
3. One of the problems that might arise with periodic scanning is that the the hardware may not be able to support simultaneous sending/receiving packets on the currently connected WLAN. For example, scanning requires the NIC to switch between different frequencies, so a NIC may not provide the support to send a probe at a given channel and listening on that channel for a reply while allowing sending/receiving packets on the currently connected WLAN as the channel is different.
4. Another issue to be considered is whether the scan intervals should be user-specifiable or not. Currently the maximum number of transmit attempts (ATH_TXMAXTRY) and the minimum scan interval (IEEE80211_INACT_MAX) both seem to be hard-coded which seems a little arbitrary. libwladm allows the user to specify properties, but it is not clear at this point whether the scan interval can be set through wladm_set_prop(). The aim is that the user should be able to increase the scan frequency via the UI, and there should also be a separate setting for the scan frequency when the system is on battery power.

4.7 Routing

Generally speaking, routing is an area where complex configuration and policy details should remain part of the subsystem, and not be subsumed by NWAM. NWAM's role should be that of administrator, specifying the circumstances under which the subsystem (or components of the subsystem) should be enabled; §4.7.1 will discuss this aspect of NWAM's relationship with the routing subsystem.

Unfortunately, it is difficult to make such a clean distinction between NWAM and the routing subsystem. There are a couple additional areas in which NWAM must interact with the routing subsystem; these are discussed in §4.7.2 and §4.7.3.

Finally, the interfaces between NWAM and the routing subsystem's Routing Information Base (RIB), for the setting of both routes and routing policy, are defined in §4.7.4.

4.7.1 NWAM as Routing Administrator

One component of the Upper Layer Profile/Environment will be the specification of which routing daemons to run. NWAM will build on the functionality introduced by the Quagga project to convert routing services to the SMF framework. As SMF services, the different routing service options can easily be enabled/disabled by NWAM as specified by the user.

The NWAM interface will offer the user all the functionality that is available through the `routeadm(1M)` command. That is, the user will be able to select whether or not IPv4 routing, IPv4 forwarding, IPv6 routing, and IPv6 forwarding are enabled, as well as which routing daemons to run if routing is enabled.

4.7.2 NWAM as Route Provider

The user may wish to set static routes for a particular environment. Thus, a list of static routes will be an additional component of the Upper Layer Profile/Environment. When the ULP/Environment is enabled, NWAM will pass these routes to the RIB, which will apply policy and send routes to the kernel's Forwarding Information Base (FIB) as needed. The interface between NWAM and the RIB is specified in §4.7.4.

4.7.3 NWAM as Routing Policy Provider

On multi-homed systems, it is possible that automatic configuration could result in multiple routes to a given destination. The most common example of this is the case where a system with two interfaces uses DHCP on both interfaces, and is given the address of a different "default" router (i.e. a route to destination 0.0.0.0 or ::0) on each interface. In that case, the user may specify a preferred interface as part of the NCU. NWAM will pass this preference to the RIB, which will take it into account when determining which routes to send to the FIB. Routes learned via DHCP and IPv6 Router Advertisements will be sent to the RIB for policy filtering, rather than directly to the kernel's FIB, as is currently the case.

4.7.4 Interfaces Between NWAM and the RIB

The zebra daemon will serve as the RIB, receiving routes from various sources (routing daemons, `dhcpagent`, `in.ndpd`, NWAM) and policy information (currently only from NWAM, but possibly from other sources in the future). Given a set of policy rules, it will decide which of the routes it receives should be passed to the FIB in the kernel; it will also have the ability to delete routes from the FIB if policy changes make this necessary.

The interface by which routes are specified will be based on the existing `PF_ROUTE` socket interface.

The interface by which routing policy is specified will be extensible, so that in the future different types of policy may be passed using the interface; but initially, interface preferences will be the only type of policy specified.

It should be noted that this design requires changes beyond the NWAM project:

- `dhcpagent` and `in.ndpd` will need to be modified to use the new interface to the RIB to set routes, rather than sending them directly to the kernel. These changes will be fairly small and can be completed by the NWAM team.
- The zebra daemon will be used as the RIB. Though this daemon currently receives routes from some routing

daemons, it will need to be extended to support both the route-setting interface and the policy-setting interface. These changes will be more involved and would ideally be completed as follow-ups to the Quagga project.

4.8 Static Configuration

Static (link & IP) configuration means everything has a fixed value, which also implies that the conditions are static and thus whatever Environment is selected is also static. This is all just a degenerate case of the rest of this document; this section is only included for the sake of completeness.

4.9 Legacy Support

4.9.1 `/etc/hostname.`*`interface`* files

One of the most vile interfaces (in the `attributes(5)` sense) in all of Solaris is using `/etc/hostname.interface` (in the `ifconfig(1m)` sense) to configure network interfaces. This project will make these interfaces obsolete, but we will still have to deal with them to be backward compatible.

Fortunately, a check of the various man pages which mention them indicates that their documented use is of the form *write to this file, then it will be consumed at the next system boot.* So migration is proposed as follows.

When the NWAM daemon starts, it will look at the existence of these files and translate the information to the SMF repository (see §4.9.2 just below for details of the SMF representation):

- If no information exists for the corresponding interface, NWAM will create an instance, `svc:/network/<TBD NWAM service name>:ip-<NCU name>`. NWAM will also need to set up appropriate dependencies between the created instance and other instances.
- If a corresponding service instance already exists, NWAM will check the information, such as property values and dependencies, and decide if there is a conflict. If not, nothing needs to be done. When there is a conflict, whatever is in the SMF repository will be honored and the user will be notified of the conflict(s).

During the lifetime of the NWAM daemon, it does not check for any changes to these files: they are only read when the service starts.

At fresh install, NWAM will automatically detect the existence of NICs, create a service instance for each interface and apply some default properties on them. If NWAM detects that some interfaces can be grouped into one IPMP group, it will do so and create the corresponding "IPMP" instance.

If the user plugs in a new device, s/he is encouraged to use the NWAM UI to configure the link and interface(s) above it. If s/he chooses to use the traditional way, editing a hostname.<if> and using ifconfig, information in the file will not be imported to SMF until a reboot or a restart of the NWAM service.

4.9.2 SMF representation

Information from `/etc/hostname.`*`interface`* files will be stored as an `:ip-<NCU name>` service instance in

SMF. More details in §5.1.

5. Network Service Model

This section might also be called "how we interact with SMF".

5.1 SMF Representation

5.1.1 Service Instances

The NWAM service can have the following instances.

:default
: This is the default instance for NWAM service, which will be brought up by svc.startd(1M). It will launch the NWAM daemon.

:link-<NCU name>
: This kind of instance represents a datalink. It will be created/modified by the NWAM daemon as needed to reflect changes to the NCP configuration. Its properties include:
  - link-class: phys, iptun, aggr or vlan
  - when to activate: whenever possible, never, or contingent
  - what kinds of address selection methods to accept on an interface above this link

:ip-<NCU name>
: This kind of instance represents an IP interface. It will be created/modified by the NWAM daemon as needed to reflect changes to the NCP configuration. Its properties include:
  - interface-class: simple, ipmp, loopback or vni
  - IP version: 4 or 6
  - IP address-source
  - when to activate: whenever possible, never, or contingent

:enm-<ENM name>
: This kind of instance represents an External Network Modifier which has been registered with NWAM. Its properties include:
  - start and stop property groups
  - Environment: the Environment to be activated when the ENM is enabled
  - state: enabled/disabled/maintenance; set by NWAM
  - when to activate: whenever possible, never, contingent, or on user request

Appropriate dependencies will be specified when an instance is created; SMF dependencies will be used to model the contingent activation of NCUs. If the value of "when to activate" is "contingent", it means whether to bring up this NCU entity is contingent to some other NCU entity. Hence a dependency should be set up between this and the other party, for example, if bge0 is up then don't bring ath0 up, then NCU-ath0 should depend on NCU-bge0 as `exclude_all` dependency. (See the Dependencies section in smf(5) for a more detailed description.)

Additionally, a dependency on each new instance will be added to the :default instance to insure that the new instance

will be included in the dependency graph. This will prevent problems caused by changing run-levels, which could disable instances that are not tied in to the graph.

There might be other dependencies. For example, a "simple" ip.<NCU name> requires a physical link under it, so it should depend on a particular link.<NCU name> as `require_all` or depend on a set of link.<NCU name> instances as `require_any`.

5.1.2 SMF profiles

Both NCPs and Environments are instances of Enhanced SMF profiles. NCPs should be one layer below Environments logically because information in NCPs is applied first and which Environment to select is determined by the NCP and the policy engine.

5.2 WEP Key Storage

The biggest concern about wepkey is its security. wepkey handle, a name to indicate a wepkey, is stored in SMF repository along with other data NWAM manipulates. The name of the wepkey should indicate not only the network this wepkey is associated but also the user who inputs this wepkey. The value of the wepkey, the secure part, is manipulated(get and set) by libdladm. When NWAM detect a wireless network which need a wepkey to access, it calls libdladm to retrieve the value of the wepkey, and use the value to connect to the network. If none is found, NWAM prompts user to input one. NWAM then uses what user inputs to connect, creates a secure object using libdladm, and sets its value to what user input for later reference.

5.3 SMF Methods

The :default instance will have a start method, a refresh method, and a stop method. The NWAM daemon will be a delegated restarter for its dynamic instances, and thus the methods for those instances will be managed internally by the daemon.

5.4 SMF Service Model

5.4.1 Proposal

The existing SMF services network/loopback, network/physical, network/initial, network/services will be deleted. The transient tasks currently done by these services will be done by the NWAM daemon except the IPsec related and the IP tunneling related parts of network/initial. (Note that one or more new services will be created for IPsec as part of the work to fix bug 6185380.)
The existence of milestone/network is misleading. Some poorly written applications may have created a dependency on it, thinking that this would ensure a connection to peer hosts. However, such a milestone cannot guarantee the connection to any peer. Ideally this milestone should be deleted. But for backward compatibility, it will will remain for now, and be marked as obsolete so that we can delete it later.

NWAM will introduce a new SMF service: network/profiled. This service will only depend on system/device/local which in turn depends on filesystem/usr. This service will have instances as described in §4.1.3.

5.4.2 How it works

First, a few definitions are needed:

- reset: bringing the system from where it is (point A) to where the active NCP says it should be (point B); there are two variants:
- hard reset: we tear everything (except manual NCUs) down, then configure everything according to the active NCP. In other words, we go from A to 0 to B.
- soft reset: we go directly from A to B, making as few changes as possible to get there.

- On boot, we configure everything according to the repository, as specified in the sequential list below. Once this configuration is complete, we place a marker somewhere (perhaps a file in `/var/run` or perhaps some other mechanism). Note that this is just a special case of the "not found" scenario immediately below; it is mentioned here merely for explanatory purpose.
- On service start, we look for this marker.
  - If it is found (i.e., we have already been running), we assume that we died prematurely, and attempt to resume subsequent operations (i.e., monitoring for new links appearing or old links going away or whatever) without making any changes as we start.
  - If it is not found (i.e., we have not already been running), we assume this is an initial start and we treat it like a hard reset, and bring anything which we find down (except manual NCUs), then configure everything per the repository (again, see the sequential list below). Note that boot is just a special case of this scenario. Note also that bringing everything (except manual NCUs) down will have the result that any network connections established on top of the affected NCUs will be lost.
- On explicit service stop, we tear everything down (except manual NCUs), including the marker. Again, note that this tear-down will have the result that any network connections established on top of the affected NCUs will be lost. To aid debugging, a committed interface will be provided to have NWAM exit without tearing anything down.
- As a result of the above, service restart behaves just like a hard reset.
- We treat service refresh like a soft reset.

Further clarification on restart: if the NWAM service were to have a dependency on some other service with a `restart_on` value of something other than `none` then there would be a risk of service disruption, but we do not anticipate any such dependencies. So the only ways the NWAM service should be administratively restarted are 1) if the user explicitly requests it via `svcadm restart` and 2) via reboot.

Distinction between restart and refresh: as mentioned above, restart behaves like a hard reset, whereas refresh is treated like a soft reset. Thus any system administrator should choose carefully between the two, as restart represents a much bigger hammer. As such, we expect that refresh will be far the more common operation, as it will not involve disrupting any existing connections or higher level service which might be using them.

The following sequential steps are taken to configure everything according to the repository:

1. NWAM plumbs and configures the loopback interface first.
2. If any security rules (IP filter and/or IPsec and/or any other security) need to be loaded when only loopback is

available, NWAM will load them. By default, IP filter will be disabled and no IP filter rules will be loaded. If the user configures any IP filter rules, NWAM can enable the IP filter start service to load them and start IP filtering.

3. NWAM then enables milestone/network.
4. NWAM applies the active NCP onto the system; i.e., certain `:link-<NCU>` and `ip.<NCU>` instances will be created.
5. NWAM then applies the stand-alone Environment to the system. Meanwhile the daemon tries to gather information from outside networks and to connect to one or more networks.
6. If some network is connected (see §2.3) and we need to switch to another Environment, the daemon will perform an Environment switch.

After starting, the NWAM daemon will keep listening for any network environment changes and user administration activities (for instance, a user asking for a profile switch or a user making a change in the active Environment). If such a change will cause an Environment switch or some service(s) to be refreshed, the NWAM daemon will do so.

5.4.3 Environments

NWAM will provide a default stand-alone Environment (applied at step 5 in §5.4.2 just above) and a default networked Enviroment. Although these two Environments will have the same default values (except for sshd), they are provided separately to facilitate customization: users can customize each of these Environments, though there should be only one stand-alone Environment. Users can create more Environments and associate each with a different network environment. Users can also choose to stay stand-alone even when a network is detected.
In any Environment, there will be a list of properties that will be applied on the system, and a list of services that will be enabled and/or disabled in that network environment.

5.4.4 Dependency rearrangement

For the services that currently have any sort of direct dependencies on those deleted services, all dependencies will be deleted. Some services will be enabled by default (meaning that NWAM profiles will not control them) and some will be controlled by an NWAM profile (enabled or disabled). Below is a detailed description of each individual services. (Note: services listed below may have other dependencies than those deleted. But since those dependencies will remain as is and to keep things simpler, they are not listed below. For instance, "network/ipfilter" depends on "network/physical", "filesystem/usr" and "system/identity:node". But since network/physical will be deleted by NWAM, we only listed the dependency on network/physical, only this dependency will be deleted and others will remain as they are.)

service name	current dependency	future default disposition
milestone/single-user:default	network/loopback (require-any)	enabled, not controlled by any NWAM profiles
system/identity:node	network/loopback (require_any)	The NWAM daemon will set the node name

system/identity:domain	and network/physical (optional_all).	and domain after it decides the value for each, then the daemon will enable the services. The services should be refreshed if the network environment changes and the change causes the nodename / domain to change.
network/ipfilter:default	network/physical (require_all).	disabled in standalone Environment
network/ssh:default	network/loopback (require_all) and network/physical (require_all).	disabled in standalone profile
network/smtp:sendmail	network/service (require_all).	enabled in standalone profile as local-only mode and needs to be refreshed if the network environment changes.
network/inetd:default	network/loopback (require_any)	enabled in standalone profile
network/iscsi_initiator:default	network/loopback (require_any)	enabled in standalone profile
network/rarp:default	network/loopback (require_any)	disabled in standalone profile
network/slp:default	network/loopback (require_any)	disabled in standalone profile
network/shell:kshell	network/loopback (require_any)	disabled in standalone profile
network/shell:default	network/loopback (require_any)	disabled in standalone profile
network/ntp:default	network/service (require_any)	disabled in standalone profile
network/dns/server:default	network/loopback (require_any)	disabled in standalone profile
network/dns/client:default	network/loopback (require_any) and network/service (require_any)	disabled in standalone profile
network/ldap/client:default	network/initial(require_all)	disabled in standalone profile

Note: if a user wants to run some disabled service(s) above, s/he can modify and provide his/her own custom copy of the standalone Environment.

6. Processes & Threads

6.1 Breakdown

NWAM will be broken down into a set of processes and threads. This section specifies how this will be done. This section is coupled with §8.1 (Internal IPC) and §8.2 (External IPC).

Along with this section are diagrams which show how the nwam daemon interacts with the system and other processes and another diagram which shows some of its internal organization into threads. The diagram is available at nwam_process.pdf.

At this time there is no room for a legend on the diagrams so here is help in interpreting the diagrams. In the first diagram (NWAM sys interaction) the diagram is broken up by a horizontal red bar which indicates the user space/kernel space split. Circles below the bar indicate conceptual sources of events. In some cases those will actually be connected through the kernel to another user space process or might actually be multiple sources of information. Boxes above the line indicate user space processes. Ovals above the line indicate shared objects or families of shared objects. A line connecting a box to an oval indicates that the process (box) uses the shared object (oval). Arrow between boxes indicate primary data flow. There is also a vertical line which indicates the separate between two zones. With shared networking stacks the configuration information will be provided by the nwam daemon in the global zone. For zone exclusive stacks the configuration information will be provided by a nwam daemon running in the zone the stack is attached to.

In the second diagram (NWAM proc design) the same kernel/user space horizontal bar exists. Below the bar are event sources. Above the bar boxes indicate threads within the NWAM process (instead of separate processes as they did on the previous diagram). Ovals indicate shared objects again. Lines indicate a uses relationship and arrows indicate data flow. Rectangles with horizontal lines through them indicate a queue used for inter-thread communication.

6.1.1 Core functionality

The core functionality of NWAM will be contained in a single daemon. Within that daemon will exist a set of threads used to manage various asynchronous needs.

6.1.2 main event loop (MEL)

A single thread will be used to manage the main event loop (section 3) The other threads will gather up events and feed to them to queue that this thread retrieves its work from.

6.1.3 event collection

A set of threads will be used to collect events. Each of these threads will collect events and place them on the main event queue for the main event loop to consume.

A single thread will be used to turn signals into synchronous events. Signals will be blocked in all threads. `sigwait(2)` is used to wait for threads and that that thread will determine how to manage that signal. This allows for the synchronous management of signals.

A thread will be used to collect file based events. At this time these are the events from the routing socket.

A thread will be used to wait for sys events.

6.1.4 Integration with Zones

Post stack instances (Nevada build 57) the administrative model for interfaces in zones with exclusive stacks is similar enough to that of a global zone and rich enough to be able to manage interfaces from within a zone. We will have a separate NWAM daemon running in the each zone to effect this. Shared stacks are administered from the NWAM daemon running in the global zone. The user can assign use of shared stacks to non global zones via zonecfg(1m).

6.1.5 Future integration with Crossbow

As more of Crossbow is integrated into Solaris the administrative model of Zones will become richer. In section 6.1 there is a description of how configuration will be applied across zones with different types of networking stacks.

6.1.6 Integration with CLI

The cli will be a single command modeled after zoneadm(1M).

6.1.7 Integration with GUI

Integration with the GUI will be done via the NWAM API defined in §7.3.

6.1.8 Integration with Gnome panel application

The API mentioned in §6.1.7 immediately above will be used to integrate with the Gnome panel application.

6.2 Internal IPC

The current design for NWAM is as a single process. Since Stack Instances integrated further work will be done to have the nwam daemons managing zone exclusive networking stacks communicating with the daemon in the global zone if necessary.

7. User Interface

7.1 Command Line Interface

See our separate page for this.

7.2 Graphical User Interface

See our separate page for this.

7.3 Data Structures & APIs

See our separate page for this.

8. Dependencies with the rest of the System

8.1 Privileges

8.1.1 Introduction

Least Privilege for Solaris (PSARC 2002/188) was introduced in Solaris 10 build 30 (2003-Feb-27). It provides a mechanism to divide up the privileges normally given to root so that processes can execute with a minimal set.

8.1.2 Process requirements

The NWAM processes will need the following privileges. Depending on how NWAM is broken into processes only a subset of these might be needed by any specific part of NWAM. The privileges needed by external processes (like dhcpagent, §2.1) are not listed here.

privilege	requirement
PRIV_SYS_NET_CONFIG	§2.3 needed to configure network interfaces
PRIV_NET_RAWACCESS	§3, 4.7 (needed for routing socket)
PRIV_PROC_EXEC	§4.3 Environment selection script
PRIV_PROC_FORK	§4.3 Environment selection script

Two types of user hook are available. The first (§4.4.2) allows the user to write code to query the state of the machine and select the appropriate environment. The second (§4.5) allows the user to provide a hook to invoke functionality not directly supported by NWAM.

The selection scripts do not need to be run with any special privileges. They are not allowed to modify their environment. This script will be run as a non-root user with basic privileges.

The configuration scripts will need to be run with privileges. Several designs have been discussed from exec'ng with all privileges to pfexec'ng from basic privileges. Since it is going to take root privileges to install one of these scripts (at least one that wants additional privileges) it is not seen as a burden for the user to have to create `exec_attr` entries for each script. Thus the configuration scripts should assume that they will be pfexec'd with a default of basic privileges.

8.1.3 Startup considerations

Nominally NWAM will be started by SMF. In order to allow an administrator to start NWAM by hand when debugging configuration privileges will be assigned to the NWAM processes (section 6) via exec_attr(4). This will be referenced via use_profile and profile in the smf_method(5) manifest.

8.2 External IPC

There are multiple external processes which NWAM must communicate with. They are listed in the following table with information about the type of information exchanged and the method of IPC.

External Entity	Purpose of	Kind of IPC interaction	Notes
RIB (Quagga)	interface preference	undefined	§4.7.1
kernel	obtain state changes	routing sockets	
DHCP	DNS server, default route, IP address, etc.	socket via libdhcpagent	
Link Status change	LINK_UP/DOWN info sys, *_RUNNING flag changes	sysevents, DLPI	
SMF	system management state changes	door via libscf	
CLI, GUI	allow for user interaction	door	§7.3
NWAM in other Zones	coordinate NWAM across zones	undefined	

8.3 Install

Between the Solaris installation program and the sysidtool(1m) suite of programs, many questions are asked of the user, several of which will no longer be needed once this project makes them obsolete. I.e., NWAM will do as its name suggests, and automatically determine network configuration for use both during installation and subsequently. If any permanent static network configuration is desired, it will be deferred; post-installation, the NWAM UI can be used for this.
Specifically, there are several programs in the sysidtool(1M) suite which will need to be modified not to ask questions for which NWAM will provide the answer. In particular:

- sysidnet will no longer need to query for network interface or IP address or IPv6 usage (IPv6 will be enabled by default)
- sysidns will no longer need to query for anything; in fact, this may go away completely
- sysidsys will no longer need to query for netmask (as no IP address will be queried for)
- sysidnfs4 will no longer need to query for the NFSv4 domain (it will be left undefined)

Note that the above may not square completely with the task breakdown claimed by sysidtool(1M), but that man page appears to be out of date; we are going thru the code to make sure we have the tasks broken down correctly. Note also that this is being addressed in a related project as part of Caiman, so consider the above changes in that context.

Looking ahead to implementation, this will require multiple putbacks, as most of NWAM will go in the ON consolidation, whereas the sysidtools are in the Install consolidation, unless Visual Panels should happen to come along soon enough to save us the trouble.

Appendix A: Glossary

Appendix B: Revision History

Revision	Date	Changes
0.1	2006-Apr-11	initial draft, cloned from Version 1.0.3 of Architecture
0.1.1	2006-Apr-18	changed section 4 name from *Configuration* to *Profiles*
0.1.2	2006-Apr-20	split sections into child pages
0.2	2006-Aug-30	re-merged child pages, updated with months of discussions
0.2.1	2006-Aug-31	updated diagram, filled in §2.2
0.2.2	2006-Sep-01	updated §2.2
0.2.3	2006-Sep-13	Added introduction to §2
0.2.4	2006-Sep-14	First draft of §4.8
0.2.5	2006-Sep-26	First draft of §5.4
0.2.6	2006-Nov-03	First draft of §4.6
0.3	2006-Nov-10	Filled in §4.1 & §4.3, merged §5.4.4 into §4.8.1, moved §5.4.5 to §4.3.3, rewrote §5.4
0.3.1	2006-Nov-21	Minor corrections to §5.4
0.3.2	2006-Nov-21	First draft of §4.3.1
0.3.3	2006-Nov-28	Filled in §4.4, minor tweaks to §4.3, added §8.3
0.3.4	2006-Dec-01	Revised §4.8
0.3.5	2006-Dec-04	First draft of §4.3.2
0.3.6	2006-Dec-05	Added §2.4
0.4	2006-Dec-08	Filled in §2.3, §6.0, §6.1, §8.1
0.4.1	2006-Dec-10	Fixed broken hierarchy in §6
0.5	2006-Dec-15	Major revision to §4.1
0.5.1	2006-Dec-19	Updated §s 4.8.2 and most of 5 to match §4.1
0.5.2	2006-Dec-21	Added §8.2
0.5.3	2006-Dec-21	Minor edits (orange text removal)

0.5.4	2006-Dec-21	Filled in §4.7, more orange text clean-up
0.5.5	2006-Dec-21	Updated §4.2 & §4.8.1 to match §4.1
0.6	2006-Dec-21	Updated §1 to match §4.1
0.7	2006-Dec-22	Inserted new §4.5 (shifting §4.5 to §4.6, etc.), updated §4.1.2, §4.3. & §5.1.1 to match
0.7.1	2006-Dec-23	Edited §8.1.2
0.7.2	2007-Jan-03	Minor edits to §s 1.3 & 1.4
0.7.3	2007-Jan-04	Minor edits to many §s
0.7.4	2007-Jan-04	Revised §2.4
0.7.5	2007-Jan-09	Revised §8.1.2
0.8	2007-Jan-11	Minor update to §4.1.3, significant updates to §s 4.5 & 5.4.2
0.8.1	2007-Jan-12	Fix typos from 0.8 updates
0.8.2	2007-Feb-02	Added §7.3: link to APIs doc
0.8.3	2007-Feb-13	Edit §6, §7.1: remove orange text
0.9	2007-Feb-16	Update §3: rewrite
0.9.1	2007-Feb-16	Update §6.1.4, §6.1.5: for stack instances
0.9.2	2007-Feb-16	Replace ASCII art in §3.1.3 with diagram
0.9.3	2007-Feb-16	Minor cleanup in §3.1
0.9.4	2007-Feb-16	Minor tweaks to §s4.1 & 5.1
1.0	2007-Feb-16	Link to CLIs page from §7.1, Design Complete
1.0.1	2007-Feb-28	Fix a number of minor issues noticed by David Bustos
1.0.2	2007-Mar-01	More minor issues noticed by David Bustos
1.0.3	2007-Mar-02	Minor issues in §5
1.0.4	2007-Mar-05	Updates due to issues noticed by David Bustos, §6.14, §8.1.2, §8.2
1.0.5	2007-Mar-05	Updates due to issues noticed by David Bustos, §2.3.3.1, §4.1.3