MOTOROLA

NetPlane Core Services Overview

User's Guide

6806800C08B

September 2007

Trademarks

Notice

Limited and Restricted Rights Legend

Contact Address

Motorola GmbH

ECC Embedded Communications Computing

Lilienthalstr. 15

85579 Neubiberg-Munich/Germany

Contents

List of Tables

List of Figures

About this Manual

Overview of Contents

This manual provides an overview on the Netplane Cores Services (NCS) software which is part of the Avantellis system software. It is divided into the following chapters and appendices.

- Chapter 1, *Introduction,* on page 13
  Provides an overview of Avantellis and its main software components
- Chapter 2, *NetPlane Core Services,* on page 17
  Descibes in more detail the components and functionality of the Netplane Core Services software which constitutes one of the most important components of the Avantellis software
- Appendix A, *NCS Toolkit,* on page 45
  Describes toolkits that ease the application development.
- Appendix B, *Related Documentation,* on page 51
  Provides references to related user manuals and standard documents

Abbreviations

This document uses the following abbreviations:

Abbreviation	Definition
AIS	Application interface specification
AMF	Application Management Framework
API	Application Programmer's Interface
AvA	Availability Agent
AvD	Availability Director
AvND	Availability Node Director
AvSv	Availability Service
CEF	Command Execution Functions
CGL	Carrier Grade Linux
CKPT	Checkpoint Service
CLI	Command Line Interpreter
CLM	Cluster Membership Service
CPA	Checkpoint Agent
CPD	Checkpoin Director
CPND	Checkpoint Node Director
CPSv	Checkpoint Service

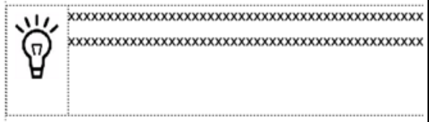
Abbreviation	Definition
CSIs	Component Service Instances
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DTA	Distributed Trace Agents
DTS	Distributed Trace Server
DTSv	Distributed Trace service
ECC	Embedded Communications Computing
EDA	Event Distribution Agent
EDS	Event Distribution Server
EDSv	Event Distribution Service
EVT	Event service
FRU	Field Replaceable Unit
FUF	Firmware Upgrade Facility
GLA	Global Locking Agent
GLD	Global Locking Director
GLND	Global Locking Node Director
GLSv	Global Lock Service
HCD	HPI Chassis Director
HISv	HPI Integration Service
HPL	HPI Adaption Private Library
HPM	Hardware Platform Manager
IfA	Interface Agents
IfD	Interface Director
IfND	Interface Node Director
IfSv	Interface Service
LCK	Locking Service
LEAP	Layered Enhancement for Accelerated Portability
LFM	Local Fault Manager
MAA	Managament Access Agent
MAS	Management Access Server
MASv	Management Access Service
MBCA	Message Based Checkpoint Agent
MBCSv	Message Based Checkpointing Service
MDS	Message Distribution Service
MIB	Management Information Base
MQA	Message Queue Agent

Abbreviation	Definition
MQD	Message Queue Director
MQND	Message Queue Node Director
MQSv	Message Queue Service
MSG	Messaging Service
NCS	NetPlane Core Services software
OAA	Object Access Agent
PCS	Platform Control Software
PSA	Persistent Store Agent
PSS	Persistent Store Server
PSSv	Persistent Store Service
SAF	Service Availability Forum™
SG	Service Groups
SI	Service Instances
SRMA	System Resource Monitoring Agent
SRMND	System Resource Monitoring Node Director
SRMSv	System Resource Monitoring Service
SSU	Simple Software Upgrade
SUND	Simple Software Upgrade Node
SU	Service Units
XML	Extensible Markup Language

Conventions

The following table describes the conventions used throughout this manual.

Notation	Description
0x00000000	Typical notation for hexadecimal numbers (digits are 0 through F), for example used for addresses and offsets
0b0000	Same for binary numbers (digits are 0 and 1)
bold	Used to emphasize a word
`Screen`	Used for on-screen output and code related elements or commands in body text
`Courier + Bold`	Used to characterize user input and to separate it from system output
Reference	Used for references and for table and figure descriptions
File > Exit	Notation for selecting a submenu
<text>	Notation for variables and keys

Notation	Description
[text]	Notation for software buttons to click on the screen and parameter description
...	Repeated item for example node 1, node 2, ..., node 12
. . .	Omission of information from example/command that is necessary at the time being
..	Ranges, for example: 0..4 means one of the integers 0,1,2,3, and 4 (used in registers)
|	Logical OR
	No danger encountered. Pay attention to important information

Summary of Changes

This manual has been revised and replaces all prior editions.

Part Number	Edition	Description
6806800C08A	February 2007	First edition
6806800C08B	September 2007	Added new section 2.9

Comments and Suggestions

We welcome and appreciate your comments on our documentation. We want to know what you think about our manuals and how we can make them better.

Mail comments to:

- Motorola GmbH
  Embedded Communications Computing
  Lilienthalstrasse 15
  85579 Neubiberg
  Germany
- eccrc@motorola.com

In all your correspondence, please list your name, position, and company. Be sure to include the title, part number, and revision of the manual and tell how you used it.

Introduction

1

1.1 Avantellis 3000 Series Overview

The Avantellis 3000 Series communications server represents the highest level of hardware and software integration to date within Motorola's family of Open Application-Enabling Platforms.

It is built on the Centellis 3000 Series hardware platform which is an AdvancedTCA system characterized by the following:

- Redundant shelf manager and alarm modules (SAM)
- Redundant switching and system controller blades (ATCA-F101)
- Redundant power entry modules (PEM)
- Various node blades incl. accessories (ATCA-7107 and ATCA-7221) verified and configured for the use within the Avantellis 3000 Series system

Avantellis 3000 Series uses this hardware platform and furthermore integrates the following software components to be used as basis for customer applications:

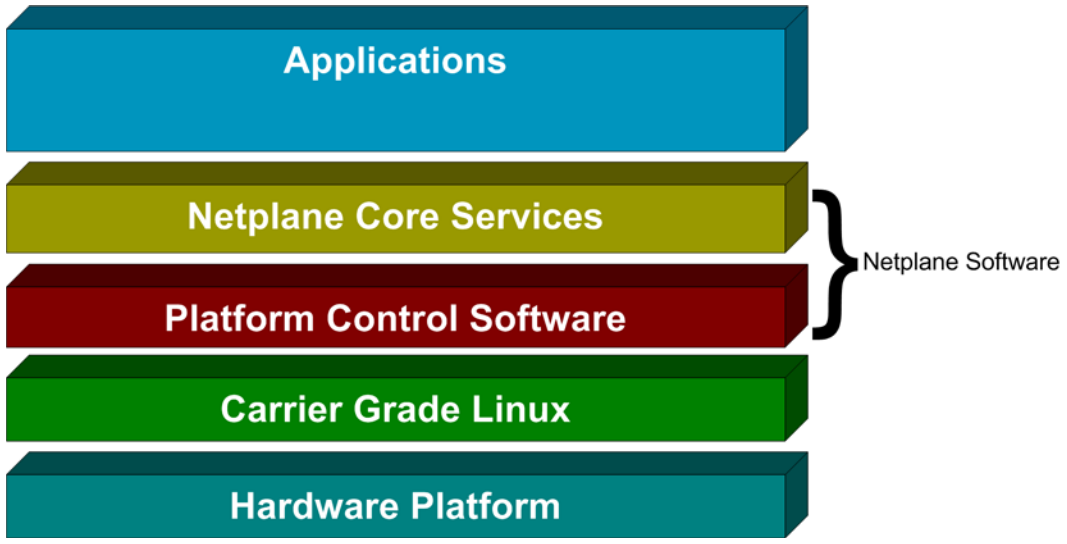
- NetPlane software
- Carrier Grade Operating System

This manual will focus on the Netplane Core Services (NCS) which is a component of the Netplane software. The purpose of this manual is only to provide an overview, information that is needed to actually use NCS for application development can be found in separate manuals. Links to these manuals are given in this manual wherever appropriate.

1.2 Avantellis 3000 Series Software Architecture

The following figure illustrates the main software components which constitute an Avantellis 3000 Series system.

Figure 1-1 Avantellis Main Software Components



1.2.1 NetPlane Software

The NetPlane software can be subdivided into the NetPlane Core Services and the Platform Control Software.

1.2.1.1 NetPlane Core Services

NetPlane Core Services (NCS) is a suite of HA middleware which implements the Service Availability Forum™ (SAF) interface specification. The SAF is a consortium of industry-leading communications and computing companies working together to develop and publish high availability and management software interface specifications.

In addition to the services that implement the SAF interface, NCS contains complementary services that were introduced by Motorola and which are required in a a complete high-availability system solution.

Both the SAF-compatible services as well as the complementary services will be described in more detail in this manual.

1.2.1.2 Platform Control Software

Platform Control Software (PCS) is an hardware and operating system abstraction layer that was defined by Motorola. It comprises software components that are specific to the underlying hardware platform. On the other hand, PCS is hardware and OS agnostic and is therefore portable across various Avantellis product lines. PCS functionality also includes software that is responsible for the HA of hardware and OS specific software entities in the system.

NCS functionality is a service user of PCS functionality. However from another perspective, PCS is a service user of NCS since the HA of PCS software components is achieved through the HA services provided by NCS.

1.2.2 Carrier Grade Linux Operating System

In an Avantellis 3000 Series system the underlying operating system is Carrier Grade Linux (CGL) V. 4.0.1.

NetPlane Core Services

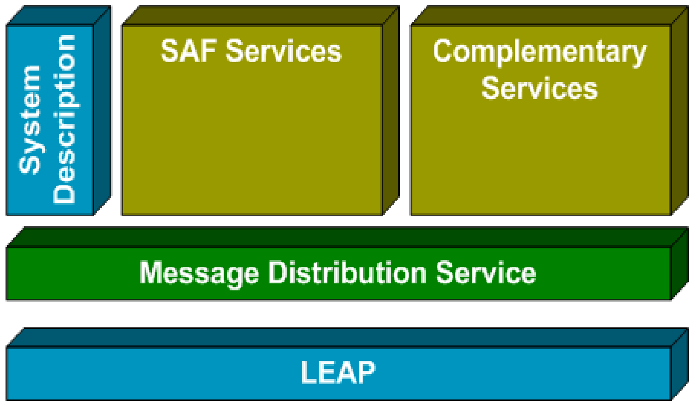
2

2.1 Architectural Overview

This chapter provides an overview on the different components of the Netplane Core Services and provides brief insights into each service.

The following figure illustrates the main software layers that constitute the NCS software:

Figure 2-1 NetPlane Core Services Components



2.1.1 NCS Services

The following table lists and briefly explains the NCS services that implement the Service Availability Forum (SAF) Application Interface Specification (AIS).

Table 2-1 SAF Compliant NCS Services

NCS Service Name	Corresponding SAF AIS Service(s)	Description
Availability Service (AvSv)	Application Management Framework (AMF) and Cluster Membership Service (CLM)	This service provides a standardized means to model system components and standardized mechanisms for monitoring, fault reporting, fault recovery and repair of components. It furthermore provides functionality that oversees cluster nodes as they join and leave the cluster.
Checkpoint Service (CPSv)	Checkpoint Service (CKPT)	This service oversees the life and integrity of entities called checkpoints. Active components write to checkpoints so that stand-by components recover the last known good state while turning active. CPSv coordinates the creation and deletion of checkpoints and maintains the checkpoint inventory within a cluster.
Message Queue Service (MQSv)	Messaging Service (MSG)	This service provides a standardized means for distributed applications to send messages among themselves. MQSv oversees entities called queues and queue groups and is capable of preserving unread messages if a reader process dies.
Event Distribution Service (EDSv)	Event service (EVT)	This service provides a standardized means to publish events and to subscribe to events anywhere in a cluster.
Global Lock Service (GLSv)	Locking Service (LCK)	This service provides a means to control access to a cluster resources by competing distributed clients.

Further details about each of these services will be given in the following sections.

The following table lists and briefly describes complementary NCS services which were developped by Motorola.

Table 2-2 Motorola-Complementary NCS Services

NCS Service Name	Description
Distributed Trace Service (DTSv)	This service allows clients to direct debug information to a file for later analysis.
HPI Integration Service (HISv)	This services allows other NCS services to access HPI events and to invoke a limited set of HPI functions.
Simple Software Upgrade (SSU)	This service facilitates node-scoped software upgrade procedures for net-booted and disk-bladed nodes.
System Resource Monitoring Service (SRMSv)	This service oversees system resource utilization by applications and notifies those applications of resource utilization events.
Persistent Store Service (PSSv)	This services tracks and makes persistent, successful changed configuration settings accomplished through the Management Access Service. The stored, persistent information may be used by a component when it is restarted.
Management Access Service (MASv)	This service provides a client with a single interface for different kinds of management operations. The supported management methods include SNMP, Command Line Interface (CLI) and XML.
Message Based Checkpointing Service (MBCSv)	This service provides message-based checkpointing between an active and one or more stand-by components.
Interface Service (IfSv)	This service oversees and coordinates the management and distribution of system-wide interface information. This includes support for virtual IP features and the MIB II interface group.

Further details about these services will be given in the following sections.

2.1.2 Message Distribution Service

The Message Distribution Service (MDS) provides high-performance, reliable message distribution services. NCS services and user applications invoke the services provided by MDS through an API which is exposed in the form of a library that can be linked to a process. MDS supports intra-process, inter-process as well as inter-node communication. Further details about MDS will be given in section 2.7 of this manual.

2.1.3 LEAP Portability Layer

LEAP stands for Layered Environment for Accelerated Portability. It is an operating system abstraction that provides portability to both NCS services and user applications from underlying operating system specifics and provides many value-added features. Further details are given in section 2.8 of this manual.

2.1.4 System Description

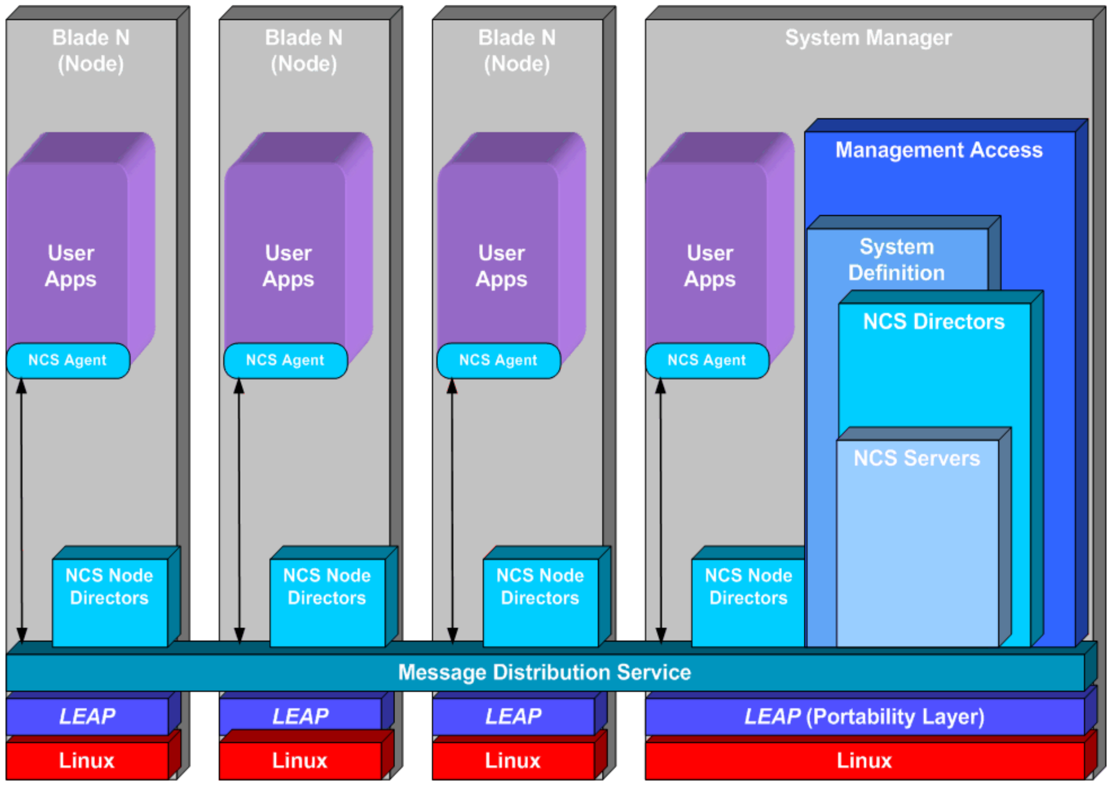
This System Description is involved in the definition of a system model. It is an XML file which defines system entities and their relation for the purpose of system configuration and management. The System Description is read in once at system start-up and from that time on stored in persistent memory.

2.2 Distribution of NCS Services in the Avantellis System

Many NCS services are subdivided into subparts. All subparts together form one NCS service. The subparts may run together on one node or may even be distributed among several nodes. This section will provide more details.

The following figure provides an overview on how NCS services are distributed in a system.

Figure 2-2 NCS Service Distribution - Overview



2.2.1 NCS Directors

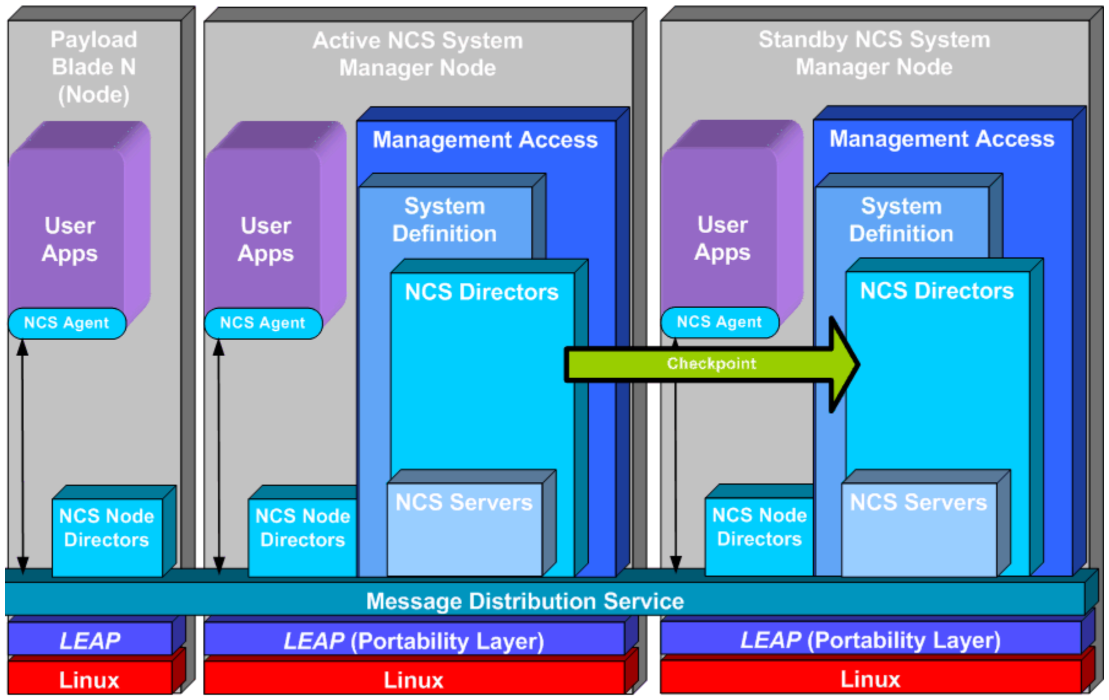
A Director for a particular NCS service manages and coordinates key data among the other distributed subparts of that service. The Director is located on a system manager node and is implemented with a 2N redundancy model.

A director communicates with Node Directors that are located on blades in a system. Node Directors handle node-scoped activities such as messaging with the central Director or with the local NCS agent.

The NCS agent makes service capabilities available to clients such as customer applications, by way of shared linkable libraries that expose well defined APIs.

The following figure illustrates Directors and their interaction with other NCS subparts.

Figure 2-3 NCS Directors



The following NCS Directors exist in an NCS system:

- Availability Director
- Checkpoint Director
- Cluster Fault Director
- Global Lock Director
- HPI Chassis Director
- Interface Director
- Message Queue Director

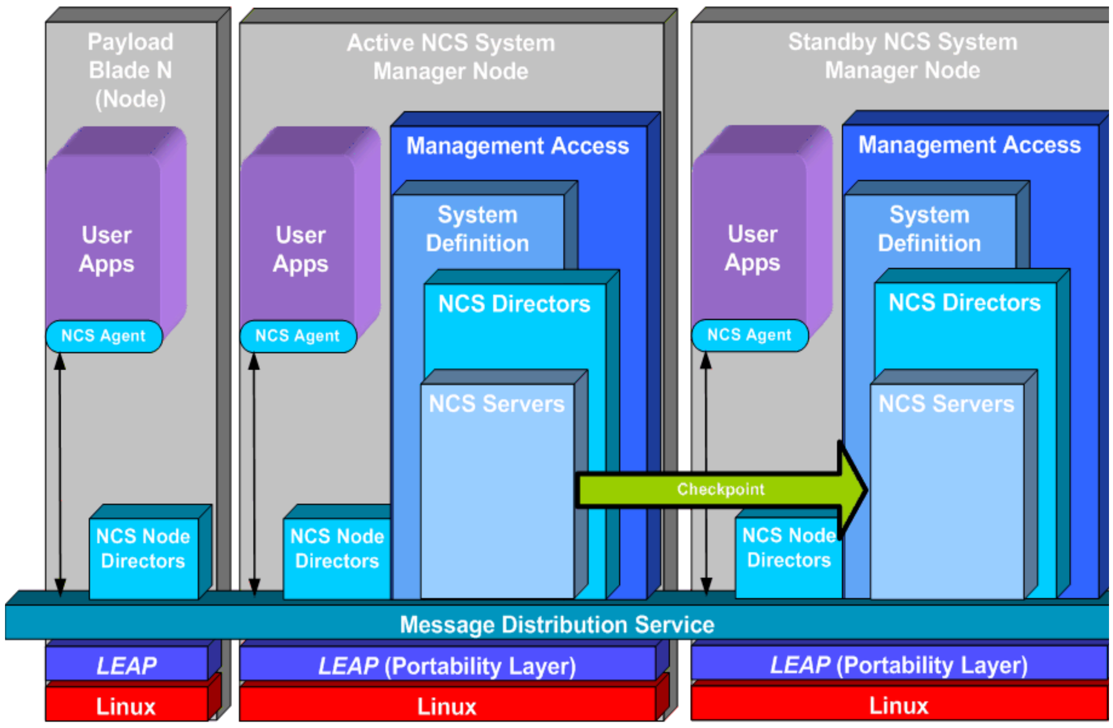
A more detailed description of these Directors will be given in the description of the respective NCS service later in this manual.

2.2.2 NCS Servers

An NCS Server provides central intelligence for a particular NCS service, but unlike with NCS Directors, there is no corresponding Node Director for an NCS Server. Instead the NCS Server communicates directly with NCS Agents.

NCS Servers are implemented in a 2N redundancy fashion. The following figure illustrates the role of NCS Servers in a system.

Figure 2-4 NCS Servers



The following NCS Servers exist in a system:

- Distributed Trace Server
- Event Distribution Server
- Management Access Server
- Persistent Store Server

A more detailed description is given together with the description of the respective NCS service later in this manual.

2.2.3 Sample Applications

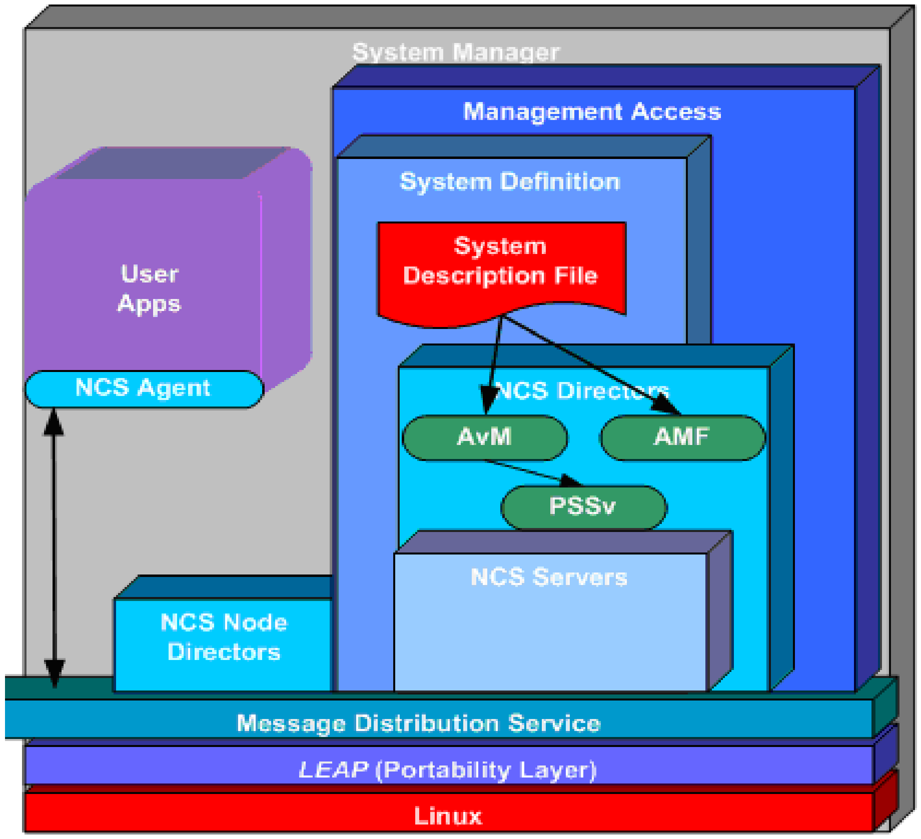
The NetPlane software suit contains a set of sample applications and sample make files that ease the development of customer applications. They illustrate the use of the various APIs and serve as a good starting-point to develop your own applications.

2.3 System Description

The System Description is an XML file which is defined prior to installation of any hardware. It describes the hardware and software configuration of a system.

The System Description file is loaded to the System manager node and read by the Availability manager during the NCS initialization. The system configuration as described in the System Description file is then stored in persistent memory (using the PSSv service) where it is subject to runtime update as a result of administrative actions via SNMP for example. The following figure illustrates how the System Description is involved in the NCS system.

Figure 2-5 System Description - Overview



The information contained in the System Description can be divided into the following categories:

- Validation description
- NCS deployment description
- Customer deployment description

The following table provides more details.

Table 2-3 System Description Content

Description Category	Description
Validation Description	This describes all possible ATCA-blade population profiles for a particular product. The description explains the inventory and legal combinations of hardware resources, primarily those hosting one or more SAF AMF nodes. Hardware resources are identified by HPI Entity Path names with key 'identifier' attributes called out.
NCS Deployment description	This identifies the actual ATCA-blade configuration and NCS Middleware deployed for this device in the network. For example, it identifies the actual blade type expected in a slot from among the 'valid blades' expressed in the Validation Description. This Description also identifies which NCS middleware pieces live where (which blade) and the redundancy models used by those NCS middleware pieces (since the NCS Middleware is itself HA). This portion of the description will likely require some adjusting at deployment time. This depends on how many plausible (and used) configurations there are for a particular product,
Customer deployment description	This identifies how customer application SW is distributed (deployed) for this device in the network. Just as with the NCS Middleware, this description identifies which application sub-parts live where (which AMF node) and the redundancy models used by those application sub-parts (as it is generally expected that customer applications shall be HA).

For further details about the System Description refer to the *System Description Programmer's Reference*.

2.4 Management Access

NCS provides two ways to access the Avantellis system for managent purposes: via CLI interface and via SNMP.

Both management access points resolve to a common managed object definition. This common definition allows the system managed objects to be distributed throughout the NCS services. These objects may be owned (implemented) by NCS, PCS and user applications as long as they have been properly integrated with the NCS Management Access Service (MASv). All NCS configuration parameters are managed by this service.

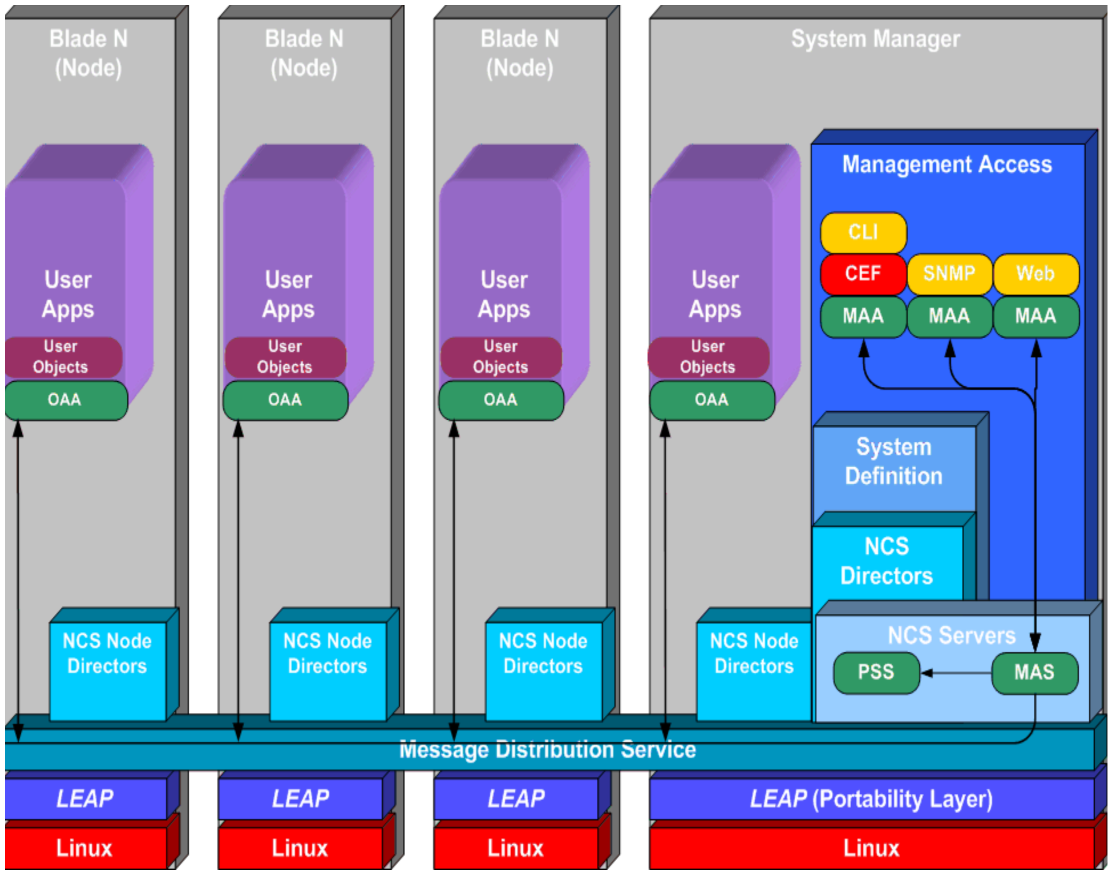
The flow is roughly as follows:

1. A MASv client, such as a CLI or SNMP Agent, translates a transaction request to a MASv format
2. The MASv client then initiates a transaction via the Management Access Agent (MAA) API library, which always forwards the transaction to the Management Access Server (MAS), where a repository of object locations is maintained.

3. The MAS then forwards the transaction to the proper object owner, wherever that happens to be in the NCS system.
4. The object owner is informed via a callback at the Object Access Agent (OAA) library APIs
5. The Object owner then does a more detailed validation check before carrying out the transaction.
6. When done (or fails), the object owner uses the OAA APIs to explain its response.
7. On the way back, if the transaction is a SET (or its proprietary variants) and successful, then the object repository overseen by the Persistent Store Server (PSS) updates the system configuration.

The following figure illustrates the communication flow.

Figure 2-6 Management Access - Information Flow



The CLI access point provides a CLI parsing system. It allows to dynamically add Command Execution Functions (CEF) to the CLI system. The CEF integrates with the Management Access Service via the Management Access Agent. The CLI is available from the console or via telnet.

The SNMP access point comprises an SNMP agent and subagent based on an open-source Linux implementation, The NCS SNMP subagent integrates SNMP requests via the management Access agent. Application-specified additional managed objects (MIBs) can be dynamically added to the subagent.

2.5 SAF-Compliant NCS Services

This section briefly describes those NCS services which implement standard SAF services. For each service an architectural overview and a functional description is given. Furthermore, a reference to user manuals is given where you can find more detailed information.

2.5.1 Availability Service

The Availability Service (AvSv) provides the following functionality:

- Leverage the SAF "System Description and Conceptual Model"
- Honour the Availability Management Framework" API
- Honour the SA Cluster membership Service API
- House the MIB tables corresponding to the hardware portion of the deployment system description which includes entity containment and fault domain hierarchy information
- House the MIB tables corresponding to the software portion of the deployment system description which include configuration of AMF-defined logical entities and their relationship
- Perform blade validation on receipt of HPI hot swap insertion events
- Handle fault events such as HPI hot swap extraction events, threshold crossing events etc.

The AvSv maintains a software system model database which captures SAF-described logical entities and their relationships to each other. The software system model database is initially configured from data contained in the System Description file. Through time the system model will modify due to changing system realities and administrative actions.

The SAF logical entities related in the system model include components which normalize the view of physical resources such as processes, drivers or devices. Components are grouped into Service Units according to fault dependencies that exist among them. A Service Unit is also scoped to one or more (physical) fault domains. Service Units of the same type are grouped into Service Groups (SG) which exhibit particular redundancy modelling characteristics. Service Units within a SG are assigned to Service Instances (SI) and given a High Availability state of active and standby.

The hardware database maintained by AvSv includes hardware entity containment information and the hardware fault domain hierarchy. All hardware entities are represented by their HPI entity paths. The hardware entity containment tree only includes managed FRUs which may or may not include processor environments., and non-FRU resources which include processor environments. The fault domain data includes dependency relationships between parent-child entities as well as non-parent child entities. The hardware system model also includes validation data for managed FRUs and the linkages between entities and AMF logical nodes. All the processor environment entities in the hardware entity containment tree, which correspond to AMF nodes, contain the node name of the associated node. The node name provides the linkage between the hardware and the software system models.

Further functionality provided by AvSv includes:

- Automatic and administrative means to instantiate, terminate and restart resources
- Automatic and administrative means to manage or reflect Service Group, Service Unit, Service Instance and Resource state
- Administrative means to perform switch-over
- Administrative means to reset (but not power cycle) nodes
- Heartbeat and event subscription schemes for fault detection, isolation and identification
- Health-check services to probe and prevent system trauma that lead to faults
- Fault recovery mechanisms to fail-over SIs which maintain service availability in case of system trauma
- Fault repair mechanisms to restore failed components
- Validation of hardware resources (managed FRUs) entering the system

The AvSv itself cannot be a single-point of failure. It provides its own internal scheme and mechanisms to protect itself from its own failure.

2.5.1.1 Architecture

The AvSv service is comprised of the following main software components:

- Availability Director
- Availability Manager
- Availability Node Director
- Availability Agent

2.5.1.1.1 Availability Director

The Availability Director maintains the most abstract portions of the Software System Model, such as cluster membership of nodes, service groups, service instances and service nodes.

Its main tasks include fault detection, isolation and recovery procedures as defined in the SAF AMF. Any problems and failures on a component that cannot be handled locally, are promoted to the Availability Director which controls and triggers the isolation of the affected component and, if possible, the activation of a stand-by component.

2.5.1.1.2 Availability Manager

The Availability Manager maintains the hardware model of the system described above. It acts as link between the underlying HPI and the AMF system modelling.

When the Availability Manager detects failures or hardware events such as extraction/insertion requests, it triggers the Availability Director to recover affected software services. When the Availability Manager receives a hot-swap event via HPI, it checks the component's FRU validation data to determine whether the component can be powered-on or not.

A further task includes reset management. When a reset request is received, the Availability Manager conducts a fault-domain hierarchy look-up. It then requests the Availability Director to switch-over the affected nodes and proceeds with the reset only if it received a confirmation from the Availability Director.

2.5.1.1.3 Availability Node Director

The Availability Node Director (AvND) resides on each system node and its main task is to maintain the node-scoped part of the software system model described above.

The AvND coordinates local fault identification and repair of components and furthermore facilitates any wishes it receives from the Availability Director.

The AvND watches for components arriving or leaving the system and summarizes this information in a Service Unit (SU) presence state, and keeps the AvD informed about the current status and changes. The AvND is capable of disengaging, restarting and destroying any component within its scope. This may occur according to AvD instructions or as a result of an administrative action or automatically triggered by policies.

2.5.1.1.4 Availability Agent

The Availability Agent (AvA) is a linkable library that exposes the SAF APIs to applications. Its task is to convey requests from the AvND or the AvD through the AvND to the application and vice versa. Details about the supported SAF APIs can be found in the *NetPlane Core Services Overview User's Guide* which is part of the Avantellis documentation collection.

2.5.2 Checkpoint Service

The Checkpoint Service (CPSv) implements the SAF Checkpoint Service. It provides checkpointing of data in a manner which is equivalent to hardware shared memory between nodes.

2.5.2.1 Basic Functionality

The CPSv maintains a set of replicated repositories called checkpoints. Each checkpoint may have one or more replicas within the scope of a cluster. At most, one replica per checkpoint may exist on one node within a cluster.

Each checkpoint comprises one or more sections which can be dynamically created or deleted. The CPSv does not encode the data written into checkpoint sections. If checkpoints are replicated on heterogeneous nodes, for example nodes with different endian architecture, you must make sure that the data can be appropriately interpreted on all nodes.

The CPSv service supports the following two types of update options:

- Asynchronous update option
- Synchronous update option

In the case of asynchronous update option, one of the replicas is designated as the active replica. Data is always read from the active replica and there is no guarantee that all the other replicas contain identical data. A write call returns after updating the active replica.

In the case of synchronous update options, the call invoked to write to the replicas returns only when all replicas have been updated, i.e. either all replicas are updated or the call fails and no changes are made to the replicas.

The CPSv supports both collocated and non-collocated checkpoints. In case of checkpoints opened with collocated and asynchronous update option, it is up to the application to set a checkpoint to the active state. In all other cases the CPSv itself handles which checkpoint is currently active.

The Checkpoint Service defined by SAF does not support hot-standby. This means that the currently stand-by component is not notified of any changes made to the checkpoint. When the stand-by component gets active, it has to iterate through the respective checkpoint sections to get up-to-date. To overcome this drawback, the CPSv provides additional, non-SAF APIs which help to notify the stand-by component of changes and thus facilitate the implementation of a hot-stand-by.

2.5.2.2 Architecture

The CPSv service consists of the following subparts:

- Checkpoint Director (CPD)
- Checkpoint Node Director (CPND)
- Checkpoint Agent (CPA)

2.5.2.2.1 Checkpoint Director

The Checkpoint Director runs on the active system manager node. Its main tasks are:

- Generating a unique ID for each new checkpoint created by applications
- Maintaining the list of nodes on which replicas of a particular checkpoint exist
- Selecting the Checkpoint Node Director (CPND) which oversees the active replica for each checkpoint
- Coordinating the creation and deletion of checkpoints
- Maintaining a repository for the CPSv policy and configuration-related information

There is an active and a stand-by CPD running respectively on the two system manager nodes. CPD uses the NCS Message based Checkpoint Service to keep the two synchronized and available for failover situations.

2.5.2.2.2 Checkpoint Node Director

The Checkpoint Node Director (CPND) runs as process both on payload blades and on the two system manager nodes. Its tasks are:

- Accepting checkpoint requests from Checkpoint Agents and streamline requests from applications to checkpoints
- Maintaining and controlling the state information pertaining to checkpoints
- Coordinating read and write accesses to/from checkpoint applications across the cluster
- Keeping track of CPNDs on other nodes in order to update the local data if a CPND that is managing the active checkpoint goes down
- Maintaining local replicas in shared memory
- Storing checkpoint control information in the shared memory so that it may be retrieved after a CPND restart
- Managing accesses to sister replicas and coordinating accesses from other applications to the replicas within the scope of the CPND

2.5.2.2.3 Checkpoint Agent

The Checkpoint Agent (CPA) is a linkable library available to applications that want to use checkpoint services.

2.5.3 Message Queue Service

The Message Queue Service (MQSv) implements the SAF Message service API.

2.5.3.1 Basic Functionality

Sender application(s) which use this service, send messages to queues and queue groups managed by MQSv and not to receiving application(s) directly. This means, if a process dies, the message persists in the queue and can be read by the restarted application or by another process.

Applications may create and destroy queues, where each queue has a globally unique name. Multiple senders may then direct messages to a queue, while a single receiver may read messages from the named queue.

Applications may group several queues into a system-wide named queue group. When sending a message to a queue group, a group policy dictates which queue actually receives the message. The sender does not know how many queues are in the group or what the policy is.

2.5.3.2 Architecture

The MQSv service consists of the following three subparts:

- Message Queue Director
- Message Queue Node Director
- Message Queue Agent

2.5.3.2.1 Message Queue Director

The Message Queue Director (MQD) runs as process on a system manager node. Its main tasks are:

- Maintaining location and state data of all queues and queue groups in a system
- Resolving all queue and queue group names and location information
- Supporting group change tracking on behalf of registering clients

There is an active and a stand-by MQD running respectively on the two system manager nodes. MQD uses the NCS Message based Checkpoint Service to keep the two synchronized and available for failover situations.

2.5.3.2.2 Message Queue Node Director

The Message Queue Node Director (MQND) runs as process both on payload and on system manager nodes. Its main tasks are:

- Managing queue send/receive operations initiated by Message Queue Agents (MQA)
- Creating, maintain and destroy queues
- Notifying MQAs when messages are delivered, received or when tracked group traits change
- Destroying a queue if its creator process dies or a retention timer expires
- Preserving messages until fetched or queue is destroyed

2.5.3.2.3 Message Queue Agent

This is a linkable library that makes all MQSv APIs available to applications.

2.5.4 Event Distribution Service

The Event Distribution Service (EDSv) is compliant with the Event Service APIs defined by the SAF.

This service controls the multiplexing of event messages in a publish/subscribe environment. It exposes a rich set of APIs which allow applications to control event distribution criteria. The implementation details of the event distribution mechanism remain transparent to the application. In the NCS environment, the EDSv uses the underlying Message Distribution Service (MDS) to implement the communication channels.

The EDSv functionality is closely linked with other NCS services, such as System Definition, Availability Service and Checkpoint Service.

2.5.4.1 Architecture

The EDSv consists of the following two parts:

- Event Distribution Server (EDS)
- Event Distribution Agent (EDA)

2.5.4.1.1 Event Distribution Server

The Event Distribution Server (EDS) is an NCS process on the System Controller blade which handles the distribution of events based on client subscriptions and filtering mechanisms. If an event was posted and event persistence was specified, the event will be retained by the server process for the time period specified in the call. During the retention time period, the EDS may redistribute the event to new subscribers for that event. Events are distributed based on a match against the filter settings specified by the subscribed client and a priority specified in the event header.

There is an active and a stand-by EDS running respectively on the two system manager nodes. EDS uses the NCS Message based Checkpoint Service to keep the two synchronized and available for failover situations.

2.5.4.1.2 Event Distribution Agent

This is a library that makes the EDSv APIs available to applications. The APIs themselves are all described in the respective SAF documents.

2.5.5 Global Lock Service

The Global Lock Service (GLSv) implements the SAF Lock Service API.

2.5.5.1 Basic Functionality

The GLSv provides a distributed locking service which allows applications running on multiple nodes to coordinate access to shared resources.

Locks are created and destroyed by applications as needed. Participating applications know that the locks exist and know how to use them. Access policies are outside the scope of the GLSv, which only provides the locking mechanism.

The GLSv supports exclusive and shared access modes. Exclusive access mode means that only one requestor is allowed through the lock at a time. Shared access mode means that multiple requestors are allowed through a lock at a time.

The GLSv furthermore supports synchronous and asynchronous APIs to carry out locking operations. In addition, GLSv provides an internal mechanism which ensures deadlock detection and prevention.

If an application creates a lock and then exits without unlocking, orphan locks are the result. Orphan locks are managed until they are properly purged from the system.

2.5.5.2 Architecture

The GLSv consists of the following subparts:

- Global Locking Director
- Global Locking Node Director
- Global Locking Agent

There is an active and a stand-by GLD running respectively on the two system manager nodes. GLD uses the NCS Message based Checkpoint Service to keep the two synchronized and available for failover situations.

2.5.5.2.1 Global Locking Director

The Global Locking Director performs the following tasks:

- Generating unique IDs referred to by an application process
- Naming one of the Global Locking Node Director (GLND) subparts as master of a particular resource
- Reelecting a new master GLND for a resource if a GLND has left the system

2.5.5.2.2 Global Lock Node Director

The Global Lock Node Director (GLND) runs as process on all the payload and system controller nodes. Its main tasks are:

- Managing the resource open and lock operation initiated by GLAs.
- For a particular resource, the GLND designated by GLD act as Master. This Master GLND is responsible for managing the lock and unlocks operations on those resources.
- GLND maintains the persistence state information in a shared memory to protect against GLND crashes and restarts

2.5.5.2.3 Global Locking Agent

A Global Locking Library (GLA) is a linkable library which makes the respective GLSv APIs available to applications.

2.6 Motorola Complementary NCS Services

This section describes in more detail the Motorola complementary NCS services which were introduced to complement the NCS services that implement SAF APIs.

2.6.1 Distributed Tracing Service

The Distributed Trace Service (DTSv) organizes, normalizes and manages logging activities among all cooperating processes within a system.

The DTSv provides the means to describe pipes that can perform operations such as read-write, filter, sort and store selectively screened logging data. The service functionality includes the creation of customized pipes as well.

2.6.1.1 Architecture

The DTSv consists of two components:

- Distributed Trace Server
- Distributed Trace Agent

2.6.1.1.1 Distributed Trace Server

The Distributed Trace Server (DTS) is responsible for defining policies based on which logs will be collected from the Distributed Trace Agents (DTA) at run-time. The logging policies can be configured via a Logging Policy MIB which is owned by the DTS. At system-start default policies will be used which can then be customized at run-time via the MIB.

2.6.1.1.2 Distributed Trace Agent

The Distributed Trace Agent (DTA) is a linkable library that makes the DTSv available to clients. The DTAs manipulate normalized logging information according to the filter descriptions configured in the Logging Policy MIB. If the Logging Policy MIB has not been configured, acceptable defaults engage. When it is configured, the DTSv selectively forwards filter descriptions to the appropriate DTAs according to the information described in the MIB. For a detailed description of the DTA refer to the *Distributed Tracing Service Programmer's Reference* .

2.6.2 HPI Integration Service

The HPI Integration Service (HISv) is a service which is only used internally by other NCS services, in particular the AvSv and the SPSv. It provides an abstraction from the actual HPI implementation used in the system. In the case of the NCS this implementation is OpenHPI.

2.6.2.1 Basic Functionality

The HISv allows its clients to interact with the underlying HPI interface. The main tasks of HISv are:

- Publishing HPI events using ESDv channels and formatting expected by subscribing clients
- Executing HPI APIs on behalf of other NCS services
- Making hardware inventory information available
- Overseeing the maintenance and manipulation of the SAF HPI MIB objects

2.6.2.1.1 Architecture

The HISv is made up of the following two components:

- HPI Chassis Director
- HPI Adaption Private Library

2.6.2.1.2 HPI Chassis Director

The HPI Chassis Director (HCD) is an NCS process which is linked to a SAF-compliant implementation of the SAF HPI library.

HCD is scoped to one chassis and there are two instances of HCD in a chassis: one active and one stand-by. The main tasks of HCD are:

- Performing HPI discovery sequence to realize the physical entities in the chassis
- Publishing HPI events
- Executing HPI APIs
- Supporting those parts of the HPI MIB which are scoped to the chassis which the HCS is responsible for

2.6.2.1.3 HPI Adaption Private Library (HPL)

The HPI Adaption Private Library (HPL) is an NCS-internal private library that is used by the SPSv and AvSv to communicate its wishes to the HCD.

2.6.3 Simple Software Upgrade

The Simple Software Upgrade (SSU) service allows to upgrade software components in the Avantellis system. This includes the upgrade of kernel, PCS software, NCS software and customer applications.

2.6.3.1 Basic Functionality

The SSU supports the upgrade of software components on system controller blades. The images are booted from disk. The upgrade itself is controlled and managed via CLI commands.

Note that for the upgrade of software components on payload blades, mechanisms like DHCP and netboot are used.

2.6.3.2 Architecture

The main component of the SSU is the Simple Software Upgrade Node (SUND). It manages the software upgrade on the respective system manager node it resides on. It furthermore handles SSU CLI commands which are issued to trigger the software upgrade.

For details about the SSU service, refer to the *Simple Software Upgrade Programmer's Reference*.

2.6.4 System Resource Monitoring Service

The System Resource Monitoring Service (SRMSv) service oversees system resource utilization by applications and notifies those applications of resource consumption events. Monitoring takes place at node-level.

The SRMSv is implemented as a bundle of System Resource Monitoring Agent (SRMA) and System Resource Monitoring Node Director (SRMND). There is no central director.

For further details about this service refer to the *System Resource Monitoring Service programmer's Reference*

2.6.5 Persistent Store-Restore Service

The Persistent Store-Restore Service (PSSv) allows to store system configuration data on a persistent storage medium. It allows to store the current configuration as well as to maintain alternate configurations.

2.6.5.1 Basic Functionality

Saving the desired configuration allows the system to initialize, retrieve the saved configuration and reach the desired state quickly, for example after a reboot. Clients may also use the PSSv to switch to another system configuration during normal operation at any point in time.

The PSSv supports the following operations and provides CLI commands for each of them:

- Listing available configuration files and display description text for each file
- Making a copy of an existing configuration file under a new name
- Renaming an existing configuration files
- Deleting an existing configuration file from the persistent store
- Displaying the description text for an existing configuration files
- Instructing the PSSv to start a new playback session from the specified configurations
- Creating a constraint spec for a configuration

2.6.5.2 Architecture

The PSSv is made up of the following components:

- Persistent Store Server
- System Description Access Manager
- System Description Parser and PSS Interactions
- Persistent Store Agent
- PSSv Target Service
- PSSv Command Execution Functions

2.6.5.2.1 Persistent Store Server

The Persistent Store Server (PSS) interacts with the other PSSv subparts and the Management Access Service. It performs the following tasks:

- When a saved configuration file is restored from the disk, the PSS fetches the configuration file from the disk and propagates the data to various components in the system using the Management Access service.
- The Management Access Service informs the PSS about configuration changes. PSS then updates its in-memory representation of the configuration file. It also forwards notifications to the System Description Parser.
- The in-memory copy of the currently active configuration is periodically stored in persistent storage.

- PSS processes set/get requests it receives from the Management Access Service.
- PSS allows the owner of a managed object to provide a template determining the method and contents of data that will be stored for that objects.

2.6.5.2.2 System Description Parser

The persistent data maintained by the PSS service includes application configuration data as well as system configuration data. The latter is stored in the previously explained XML-based System Description file. When the system powers up, the System Description Parser reads the System Description file and parses it. It then primes the system with the system configuration data.

Whenever the System Description file was changed (for example by an administrator to reflect hardware changes in the system), the System Description Parser parses the new System Description and reconciles the differences between the current system configuration and the configuration specified by the System Description.

2.6.5.2.3 Persistent Store Agent

The Persistent Store Agent (PSA) resides in NCS services which use the services of the PSSv. The PSA is responsible for collecting responses to SET/SETROW requests and forwarding that information to the central PSS. For further information about the PSA refer to the *Persistent Store-Restore Service Programmer's Reference* .

2.6.5.2.4 PSSv Command Execution Functions

The PSSv Command Execution Functions (PSSv CEF) are registered with the NCL CLI. They are invoked when commands are issued at the console. PSSv CEF supports a PSSv set of managed objects and integrates with the Management Access Service.

2.6.6 Management Access Services

The Management Access Services (MASv) provides clients with a single interface to access distributed managed objects that may be located anywhere in the system.

2.6.6.1 Basic Functionality

The MASv provides the following functionality/mechanisms:

- Common method of converting access to managed objects from any input source to a standard form
- Method of distributing ownership of the managed objects in a system to all the service and application processes in a system
- Service for locating the ownership of any particular object within the system
- Command Line Interpreter which supports all NCS services and which can be customized to support any additional customer-defined commands

- SNMP subagent which integrates the MASv with an SNMP agent and which can be configured to support any additional customer-defined managed objects
- Interface to PSSv for non-volatile storage of the values of all managed objects

MASv provides a normalized structure for the managed objects of a system. Although this structure is NCS-specific in many aspects, in many cases it uses a table indexing that is identical to that used in SNMP standard MIBs.

2.6.6.2 Architecture

The MASv consists of the following subparts:

- Management Access Agent
- Object Access Agent
- Management Access Server
- Command Line Interpreter Management Access Point
- SNMP Management Access Point

2.6.6.2.1 Management Access Agent

The Management Access Agent (MAA) provides the link between management input services, such as SNMP or CLI agent, and the common MASv services.

Applications or NCS services can choose between the following two forms to receive responses from management access requests:

- Asynchronous function callback
- Synchronous function call return

For further details refer to the *Management Access Service Programmer's Reference*.

2.6.6.2.2 Object Access Agent

The Object Access Agent (OAA) provides an interface which allows applications and NCS services to access managed objects. In particular, NCS services and applications can register the following information:

- Ownership of an entire managed object table
- Ownership of a set of scalar managed objects or a row (or range of rows) of a managed object table
- Withdrawing ownership claims regarding scalars, table rows, or an entire table
  The NCS service or application can then perform read/write operations on an object or a row of objects.

2.6.6.2.3 Management Access Server

The Management Access Server (MAS) manages the database of managed objects. It performs the following tasks:

- Receiving managed object row ownership information from the OAAs
- Direct management access requests from MAAs to the proper OAA

2.6.6.2.4 Command Line Interpreter Management Access Point

The Command Line Interpreter (CLI) provides a command-line based user interface for the NCS system. It provides a mechanism for describing commands that can be used to manage configurable entities. The commands are customizable depending on the needs of the application or NCS and are dynamically registered and deregistered. The functionality provided by the CLI includes:

- Displaying of available commands and results
- Programmatic description of command syntax and semantics
- Menu navigation
- Key assignment properties (hot keys)
- Dynamic registering of command descriptions
- Fetching character stream from input sources such as a file or terminal
- Matching character stream against all possible command descriptions and when a match is found invoke respective function

The CLI provides an API which allows applications or NCS services to register and deregister CLI commands. A detailed description of this API can be found in the *Command Line Interface Programmer's Reference*.

2.6.6.2.5 SNMP Management Access Point

The SNMP Management Access Point is an SNMP subagent whose purpose is to provide a mapping between the SNMP management agent and the MASv. This mapping allows applications and NCS services to access managed objects maintained by applications and NCS services. The main tasks of the SNMP Management Access Point are:

- Registering MIB rows with the SNMP master agent
- Receiving and forwarding SNMP requests from the SNMP master agent

For further details refer to the *SNMP SubAgent Programmer's Reference*.

2.6.7 Message-Based Checkpointing Service

The Message-Based Checkpointing Service (MBCSv) was introduced to complement the SAF-compliant Checkpointing Service. MBCSv defines a simple synchronization protocol which keeps an active and one or more of its stand-by entities in synchronization. Unlike the SAF-compliant Checkpointing Service, the MBCSv does not maintain replicas.

2.6.7.1 Basic Functionality

The main tasks of the MBCSv are:

- Dynamic discovery of peer entities
- Providing an interface to the active entity for checkpointing the state updates to the stand-by peers
- Whenever the clients's role changes to stand-by from any other role or whenever a new active client is detected, the client synchronizes its state with that of the active client (cold synchronization)
- Periodic synchronization of the client's state with that of the currently active client to obtain an abbreviated summary account (warm synchronization)
- Driving client behavior depending on the HA role assigned by the client application

2.6.7.2 Architecture

The only component of the MBCSv is the Message-Based Checkpointing Agent (MBCA). It provides stateful, message-based checkpoint replication services for its clients. For more details about the MBCA refer to the *Message Based Checkpointing Service Programmer's Reference*.

2.6.8 Interface Service

The Interface Service (IfSv) provides a common means of configuring physical and logical interface information in the NCS system, and allowing that information to be distributed to all applications and services within the system.

2.6.8.1 Basic Functionality

The IfSv provides a means by which applications and NCS services can delete and modify interface information for which they are responsible.

The IfSv supports the MIB II Interfaces group of managed objects as well as some Motorola extensions. Furthermore IfSv supports the RMON-MIB defined in RFC 2819 and RMON2-MIB defined in RFC 2021.

The IfSv interfaces with an ifIndexAlocator Platform Service in order to define an ifIndex value for each interface created. If this service is not provided, IfSv generates an ifIndex using an internal mechanism.

2.6.8.2 Architecture

The IfSv consists of the following components:

- Interface Director
- Interface Agents

2.6.8.2.1 Interface Director

The Interface Director (IfD) is located on the system manager node. Its main functions include:

- Defining an ifIndex for each configured interface (either obtained from ifIndex Allocator Platform Service or generated internally)
- Responding to requests from Interface Agents to provide an ifIndex
- Registering ownership of the interface managed objects
- Maintains a consolidated database with respect to the interfaces present on all the nodes.

2.6.8.2.2 Interface Node Director

The Interface Node Director (IfND) exists on each node. It is responsible for state management of the physical interfaces, including status monitoring. The interfaces of the IfND are private and internal to NCS.

2.6.8.2.3 Interface Agents

Interface Agents (IfA) are a linkable library that provides the IfSv functionality to its clients. Its functionality includes:

- Providing an interface to applications which they use to create and delete physical and logical interfaces
- Fetching an ifIndex for each created interface from the IfD
- Registering ownership of the managed object rows for the interfaces created
- Providing an interface that allows applications to register with the IfSv
- Notifying applications of any operational state changes of an interface
- Providing interface information to applications

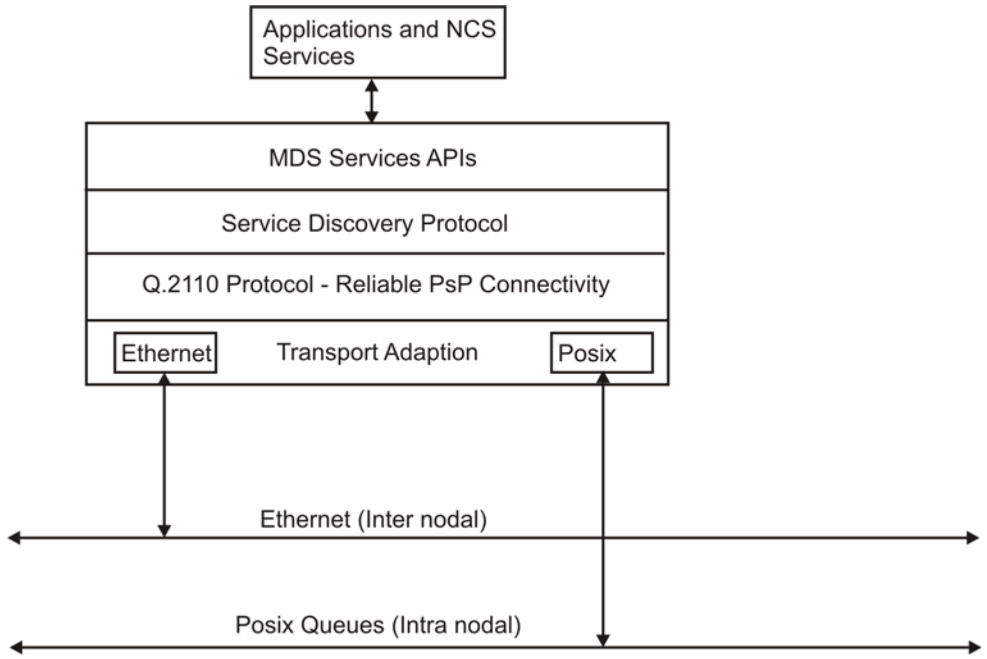
For further details refer to the *Interface Service Programmer's Reference*.

2.7 Message Distribution Service

The Message Distribution Service (MDS) provides a communication network to be used by threads and/or processes distributed over physical nodes in a system. MDS clients can dynamically find and talk to other MDS clients anywhere in the same MDS system.

The following figure provides an overview of the components that constitute the MDS.

Figure 2-7 Message Distribution Service Software Components


Applications and NCS Services
MDS Services APIs
Service Discovery Protocol
Q.2110 Protocol - Reliable PsP Connectivity
Ethernet
Transport Adaption
Posix
Ethernet (Inter nodal)
Posix Queues (Intra nodal)

The MDS Services API make the MDS available to its clients, i.e. other NCS services, customer applications or ECC applications. For a detailed description of the API, refer to the *Message Distribution Service Programmer's Reference*.

The Service Discovery Protocol layer realizes the connection paths between all MDS participants. If connections are broken or connected or participants die, fellow MDS instances are informed asynchronously through callbacks which other services are no longer reachable.

MDS uses the Q.2110 Reliability P2P as underlying communication protocol. It has been adapted slightly to NCS needs and provides:

- Transfer of data with sequence integrity
- Error connection by selective retransmission
- Error reporting to layer management

The Transport Adaption layer is an abstraction layer which hides the details of the underlying transport protocol. In the Avantellis system Ethernet is used for inter-nodal base fabric communications and POSIX queues are used for intra-nodal communications.

2.8 LEAP Portability Layer

The LEAP (Layered Enhancement for Accelerated Portability) layer was introduced to facilitate the porting of NCS applications and user applications to other operating systems. It provides various abstractions from OS-specific details. In particular, LEAP provides:

- IP-layer abstraction
- File system abstraction
- Dynamic library abstraction
- Monitored buffer and memory manager
- Monitored process internal queues
- Monitored timer services which can oversee a large number of timers at once
- Object locking

LEAP provides its services in the form of C macros and libraries to be linked by applications. For further details on the LEAP and on how to use it, refer to the *LEAP Programmer's Reference*.

2.9 Implementation Notes

This section summarizes important information that should be kept in mind when writing applications that make use of NCS.

2.9.1 Cancelling Application Threads

NCS libraries is not a cancel-safe implementation. If an application thread acquires a NCS resource (Example: acquired a lock, waiting on semaphore, etc) then the thread should be cancelled only after releasing the resource acquired by the thread. Similarly, the application thread should not be cancelled when execution of NCS API is under progress.

NCS Toolkit

A

A.1 Introduction

The NCS software is accompanied by a toolkit that enables you to develop NCS-based applications. This section describes the toolkit and its functions.

A.2 Toolkit Installation

The NCS toolkit is a tar file with the naming convention: `ncs<release name>_dev_<build name>.tar.gz`. (for example: `ncs06A_dev_3.1.4.tar.gz`, where the release name is "06A" and the build name is "3.1.4"). The tar file is packaged to be installed on a SuSE 9.0, i386 platform.

You can extract the tar files in the directory of the your choice <install-dir>. The command to extract the tar is: t**`ar -zxvf <tar file name>.`**

This creates the directory `ncs<release name>_dev_<build name>` in the location in which it is installed (`<install dir>`).

On extracting this tar file, all files related to the toolkit are installed in the directory `<install-dir>/ncs<relase name>_dev_<build name>/`. Name this as TOOLKIT_HOME directory.

To uninstall this version of the toolkit, simply delete the directory, ncs<release name>_dev_<build name>.

A.3 Toolkit Contents

The toolkit contains the following components:

- NCS software headers
  These prototype the routines implemented by NCS software. They are located in the `$(TOOLKIT_HOME)/include` directory
- NCS libraries
  NCS applications should be linked with these to use NCS Services. The libraries are located in the `$(TOOLKIT_HOME)/libs` directory, which has one directory per target. For example, `$(TOOLKIT_HOME)/libs/linux-ppc` contains NCS libraries built for a Linux PowerPC architecture

- NCS MIBs
  These are implemented by NCS services and are used to configure them. They are located in the `$(TOOLKIT_HOME)/source /mib` directory.
- NCS sample programs and makefiles
  These are provided to jump-start the application development process by providing code that is illustrative of the usage of NCS.The sample programs also serve to demonstrate the basic capabilities of each service. Application developers can leverage the makefiles provided with the sample code to build their development setup for cross-compiling the application for all targets supported by NCS
- NCS tools
  These help develop NCS-based applications. They include an SMIDUMP tool used for generating code automatically to help implement MIBs

A.4 Building the Samples

This section describes how NCS sample programs can be built for different targets supported by NCS.

A.4.1 Development Host Prerequisites

The development host should be based on the SuSE Linux 9.0 (Intel i386 architecture).

It should also have the following build support installed:

- MVL cross-compile tool-chain, which is required to build sample applications for a real NCS target.
- SuSE Linux 9.0 native, which is required to build sample applications for a development NCS target. (A development target is one in which some NCS target interfaces are provided by dummy plug-ins.)
- NetSNMP 5.1.1 headers available in the (cross) compilation before building the “snmptm_demo” product. (Please refer to the *Management Access Service Programmer’s Reference* for details.)

A.4.2 "Make" Commands

This section describes a script that allows to build the sample applications.

Description

The script described here allows to build sample applications that are available for most NCS services. The script is located in: `$TOOLKIT_HOME/source/common` and it has to be invoked from a BASH shell command prompt.

More service-specific build instructions, if any, are defined in the respective user’s guides. The generated demo binaries and libraries are in the `$TOOLKIT_HOME/source/common/bin/<target-architecture>` and `$TOOLKIT_HOME/source/common/libs/<target-architecture>` directories respectively.

Synopsis

```
./make_env.sh  <target-architecture>  <product>
```

Parameters

`<target-architecture>`

The parameter target-architecture determines the CPU architecture of the target. It can be one of the following:

- `mvl-ppc`
  PowerPC-based target
- `mvl-i686`
  Intel(c) Xeon-based target
- `mvl-em64t`
  Intel (c) 64-bit Xeon (LV-Nocona)-based target
- `linux-i386`
  Intel(c) i386-based target (meant for a development target)

`<product>`

This parameter determines which of the sample applications for the different NCS services is to be built. The following values are possible.

- `leaptest_demo`
  Builds the LEAP sample program
- `edsv_demo`
  Builds the EDSv sample program
- `cli_demo`
  Builds the CLI sample program
- `glsv_demo`
  Builds the Global Lock Service sample program
- `ifsv_demo`
  Builds the Interface Service sample program
- `cpsv_demo`
  Builds the Checkpoint Service sample program
- `mqsv_demo`
  Builds the Message Queue Service sample program
- `dlsv_demo`
  Builds the Distributed Tracing Service sample program
- `avsv_demo`
  Builds the Availability Service sample program
- `snmptm_demo`
  Builds the SNMP TestMib sample application program

- `mds_demo`
  Builds the MDS sample program
- `mbcsv_demo`
  Builds the Message-based Checkpointing Service sample program
- `srmsv_demo`
  Builds the System Resource Monitoring Service sample program

Example

To build the LEAP sample application for a PowerPC target, run the following command:

```
./make_env.sh  mvl-ppc  leaptest_demo
```

A.5 Running the Sample programs

This sections describes the steps to run the sample programs.

A.5.1 Target Prerequisites

The sample programs can only be run on the target architecture they have been built for. Therefore, a sample program first needs to be built based on the target's architecture. Refer to Appendix A, *Building the Samples* for steps to cross-compile a sample program for a target.

The target could be a system manager host or a payload node, based on the design of the sample program.

After building the sample programs, executable (binaries) are created in the `$TOOLKIT_HOME/source/common/bin/<target-architecture>` directory. Copy them from the development host to the target where they are intended to run. The target directory can be of your choice, however, we recommend `/opt/motorola/ncs/scxb/sample_programs or /opt/motorola/ncs/pld/sample_programs`.

Sample program may need additional files to be copied onto the target. Please refer to the respective user's guides for them.

Make sure that NCS software for a particular target is installed on the target machine. The configuration files should be edited to run the NCS software correctly. Most of the samples require the NCS services to be running on the target. The existence of a `/opt/motorola/ncs/scxb` indicates that the target is a system manager host and the existence of a `/opt/motorola/ncs/pld` indicates that target is a payload node.

If the target for the sample program is a payload node, the system manager host software should also be running on a connected node.

A.5.2 Setting LD_LIBRARY_PATH

Set the environment variable LD_LIBRARY_PATH to include the directory in which NCS libraries are installed.

For a system manager host setup, the path is: `export LD_LIBRARY_PATH=/opt/motorola/ncs/scxb/lib`

For a payload node setup, the path is: `export LD_LIBRARY_PATH=/opt/motorola/ncs/pld/lib`

For running the AvSv demo, the Xerces XML parser must be installed and available. The Xerces package should be installed in `/opt/xerces-c-src_2_5_0`. The library path for the package, `/opt/xerces-c-src_2_5_0/lib`, must also be appended to the existing LD_LIBRARY_PATH environmental variable. If the package is installed in some other location, the equivalent library path should be appended to the existing LD_LIBRARY_PATH environmental variable.

A.5.3 Running the Sample Programs

After setting the LD_LIBRARY_PATH, the sample programs can be run from the location where they have been copied (from the development host) on the development target. For a description of how to execute these sample applications, please refer to respective user's guides.

For example: To verify the LEAP sample application, execute the leaptest_demo.out binary. For options, refer to the LEAP Sample Application section in the LEAP Programmer's Reference.

Related Documentation

B

B.1 Motorola Embedded Communications Computing Documents

The Motorola publications listed below are referenced in this manual. You can obtain electronic copies of Embedded Communications Computing (ECC) publications by contacting your local Motorola sales office or by visiting ECC's World Wide Web literature site: http://www.motorola.com/computer/literature. This site provides the most up-to-date copies of ECC product documentation.

Table B-1 Motorola Publications

Document Title	Publication Number
Availability Service Programmer's Reference	6806800C44
Avantellis 3000 Series Rel. 3.0 User' s Guide	6806800B91
Checkpoint Service Programmer's Reference	6806800C47
Command Line Interface Programmer's Reference	6806800C11
Distributed Tracing Service Programmer's Reference	6806800B40
Event Distribution Service Programmer's Reference	6806800C48
Global Lock Service Programmer's Reference	6806800C49
HPI Integration Service Programmer's Reference	6806800C51
Interface Service Programmer's Reference	6806800B50
LEAP Programmer's Reference	6806800B56
Management Access Service Programmer's Reference	6806800B55
Message Based Checkpointing Service Programmer's Reference	6806800B41
Message Distribution Service Programmer's Reference	6806800B89
Message Queue Service Programmer's Reference	6806800C50
NetPlane Core Services Overview User's Guide	6806800C08
Persistent Store Restore Service Programmer's Reference	6806800B54
Simple Software Upgrade Programmer's Reference	6806800B19
SMIDUMP Tool Programmer's Reference	6806800B37
SNMP SubAgent Programmer's Reference	6806800B38
System Description Programmer's Reference	6806800B90
System Resource Monitoring Service Programmer's Reference	6806800B39

B.2 Related Specifications

For additional information, refer to the following table for related specifications. As an additional help, a source for the listed document is provided. Please note that, while these sources have been verified, the information is subject to change without notice.

Table B-2 Related Specifications

Document Title	Version/Source
Service Availability Forum Application Interface Specification, Volume 1, Overview and Models	SAF-AIS-B.01.01/ http://www.saforum.org
Service Availability Forum Application Interface Specification, Volume 2, Availability Management Framework	SAF-AIS-AMF-B.01.01/ http://www.saforum.org
Service Availability Forum Application Interface Specification, Volume 3, Cluster Membership Service	SAF-AIS-CLM-B.01.01/ http://www.saforum.org
Service Availability Forum Application Interface Specification, Volume 4, Checkpoint Service	SAF-AIS-CKPT-B.01.01/ http://www.saforum.org
Service Availability Forum Application Interface Specification, Volume 5, Event Service	SAF-AIS-EVT-B.01.01/ http://www.saforum.org
Service Availability Forum Application Interface Specification, Volume 6, Message Service	SAF-AIS-MSG-B.01.01/ http://www.saforum.org
Service Availability Forum Application Interface Specification, Volume 7, Lock Service	SAF-AIS-LCK-B.01.01/ http://www.saforum.org