

Chapter 1. TSP Introduction 3

- TSP System 3
- TSP System Overview 5
- Linux In The Cluster 7
- TelORB 9
- More TSP Features 11
- Software Architecture 12
- HARDWARE 14
- Configurations 18
- Processes 21
- Distribution controlled by CM 26
- Distribution Concepts 26
- Distribution of processes 27
- Distribution of database instances - POTs 28
- Communication 32
- The Database (DBN) 36
- High Availability Supported by TelORB Clusterware 39
- Software Upgrade 41
- Node Management 43
- OAM Toolbox 43
- TADE 45
- From Design to Deployment 47

Chapter 2. Configuring the Kickstart Server 49

- Kickstart Server Installation Instructions 49

Chapter 3. Configuring the IOs 64

- Minicom on the Kickstart Server 64
- Installing Linux on IO 1 64
- The roles of the IO processors 65
- Site-specific Scripts 69
- Loading the Cluster 73
- Start phases 74
- Initial Start 76
- Different types of start - restart 93

Chapter 4. Configuring the Traffic Processors 96

- Specifying the site configuration in the epct file 99

Chapter 5. Applying the Configuration 120

- LoadingGroups 120

Chapter 6. Software Upgrade 125

- Introduction 125
- Software Upgrade 125
- The upgrade package 127

Chapter 7. Exercises 130

- TSP Introduction 131
- Configuring the IO Processors 136
- Installing the IO Processors 139

Examining the IO Processors 148
Building the SiteConfiguration 153
Checking the BIOS of Traffic Processors 156
Loading the SiteConfiguration 157
Transfer the new loadingGroups 159
Examining the System 162
Loading an application as a Software Upgrade 165
TelORB Traffic Simulator 166
Overview 167
Printouts 172

TSP Introduction

TSP (The Server Platform) is an Ericsson carrier-class server technology designed for server solutions where high availability is required. Ericsson - besides evolving AXE - has introduced two carrier-class technologies:

- TSP for servers
- CPP for gateways

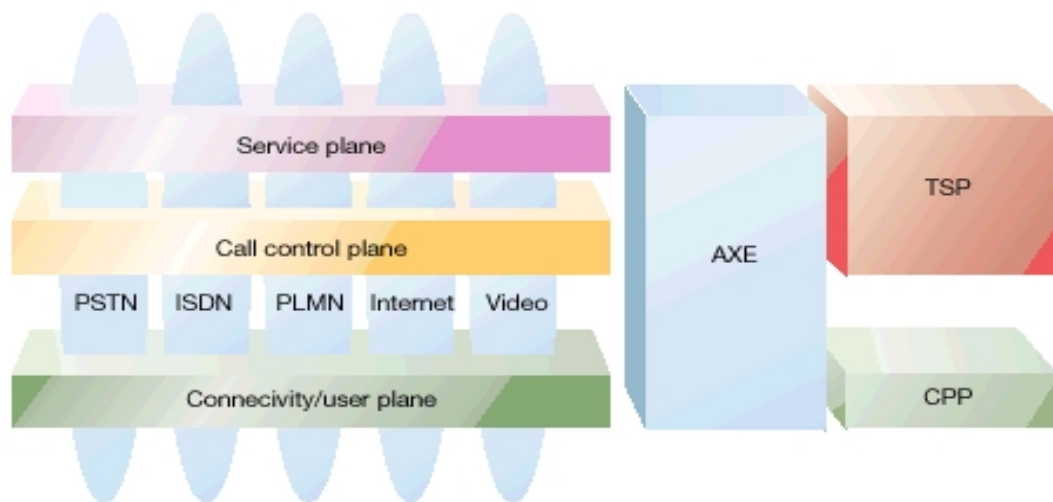


Figure 1. TSP in the layered network

TSP System

Some of the characteristics of TSP are:

- Support for geographical network redundancy
- Support for online SW upgrades
- Redundant HW for fault tolerance.
- Use of standardised HW platforms
- Object orientation
- High system availability
- Scalability – where addition of processor boards increase capacity linearly

Examples of TSP applications include:

- home location register (HLR),
- the media gateway control function (MGCF),
- multimedia servers (SCS),

- authentication, authorization and accounting (AAA) servers.

These nodes, typically located in the center of a network, must provide superior inservice performance and must be allowed to grow in capacity. They include central databases that contain essential user data, traffic data and charging data.

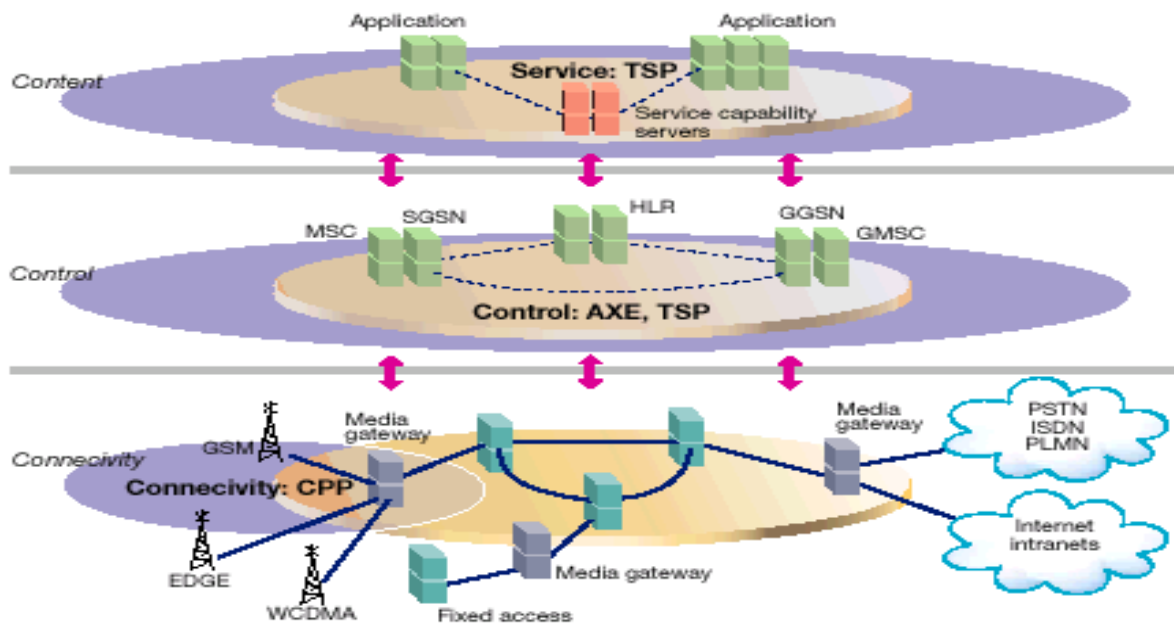


Figure 2. TSP Applications in the network

TSP uses commercially available components. Only the most essential components for adding value in terms of robustness and scalability are developed by Ericsson.

TSP System Overview

The following picture (see fig. below) gives us an overview of the TSP system. There are a number of processors, interconnected through ethernet switches.

Essentially, there are two kinds of processors:

- IO-processors - The two IO processors run LINUX. They are used for TelORB specific I/O operations like loading, upgrades and taking of backups
- Traffic processors (TP's) - Traffic processors are used to execute the applications. From TSP 4 there is an option to use either the TelORB specific operating system Dicos or LINUX for the traffic processors. The incoming traffic arrives as IP or SS7 calls via traffic processors with special interfaces for these types of traffic.

The TelORB processors and the IO processors are connected to two different LAN's (see figure below). If a communication failure on one LAN should occur, the system is still fully functional.

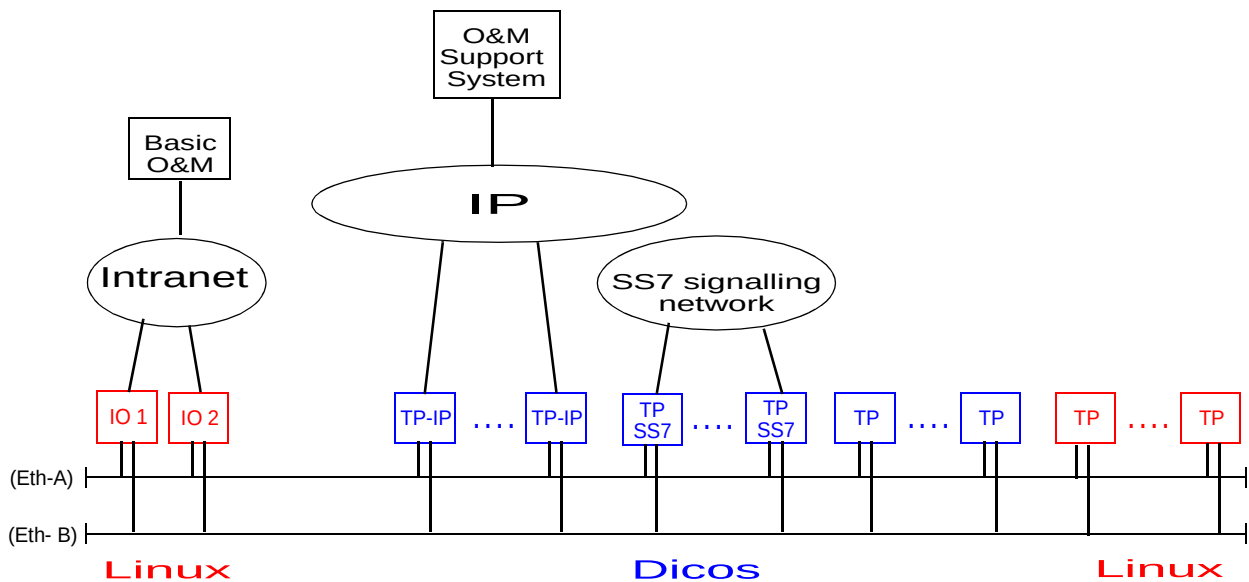


Figure 3. TSP System

Loading Software - IDPs

The software of a TSP system comes as a set of *Internal Delivery Packages*, IDPs, that contain one or more *Load Modules*, LMs. A load module contains the code to be actually executed on the Traffic Processors.

Site Configuration

A site configuration is built by using the epct command. The epct files describe the HW and SW configuration of the site.

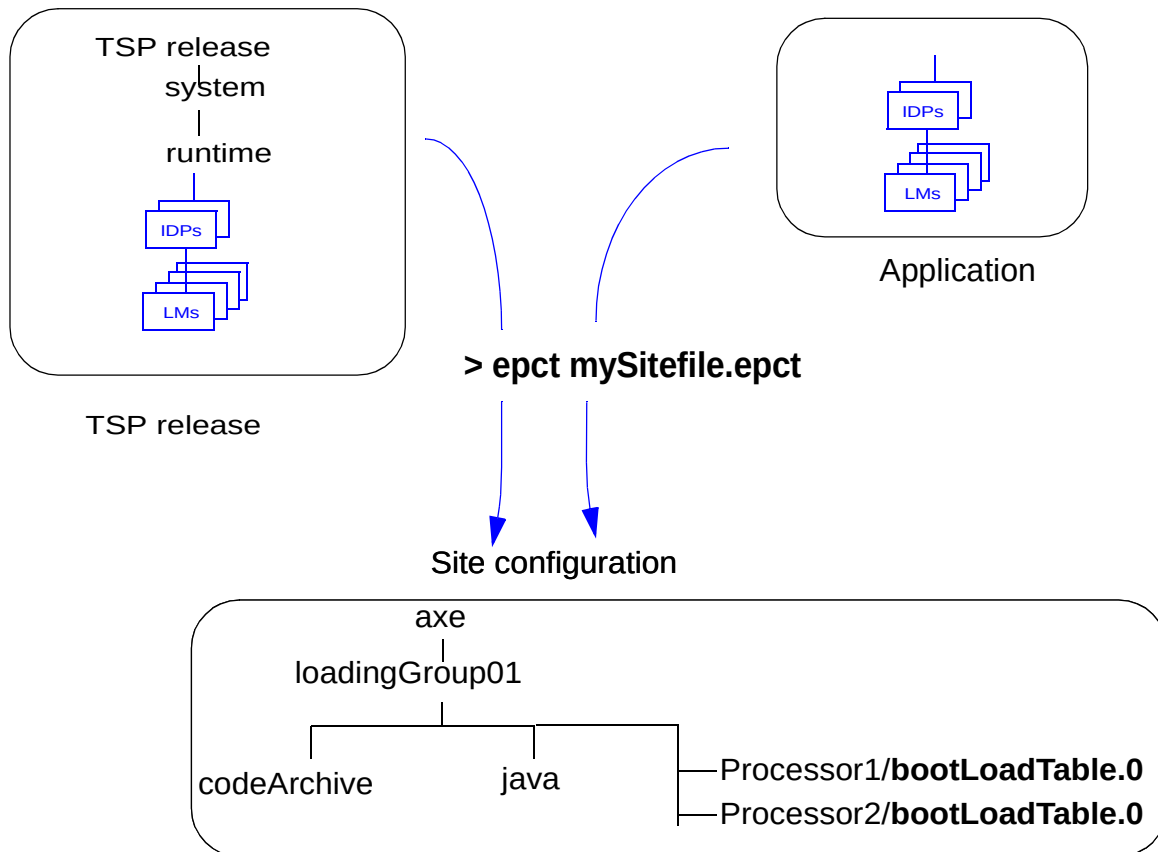


Figure 4. Building a site configuration

Before the TSP system is started for the first time, the generated site configuration is transferred to the two LINUX computers called the I/O processors (see figure below). The Traffic Processors are empty. Some of the loadmodules are part of TelORB itself, others contain application code.

When the system is booted the loadmodules are transferred from the IOs onto the traffic processors in a multi-cast load. The TSP loadmodules are loaded during the earlier start phases, application loadmodules are normally loaded later.

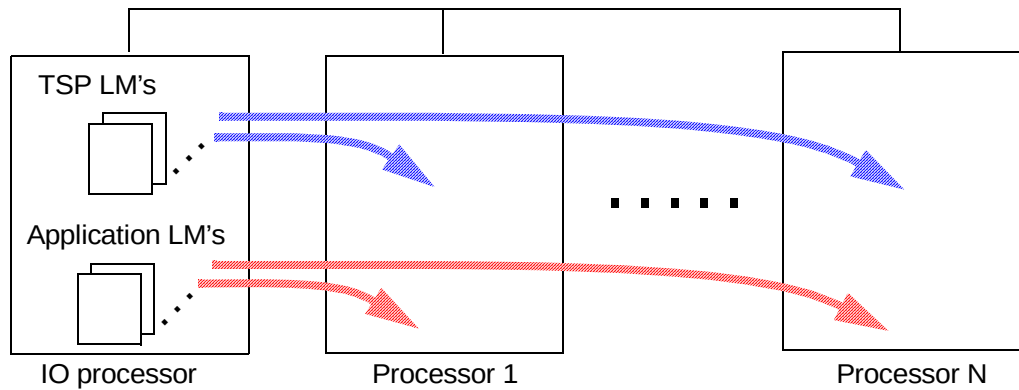


Figure 5. The processors are loaded with loadmodules from the I/O processor.

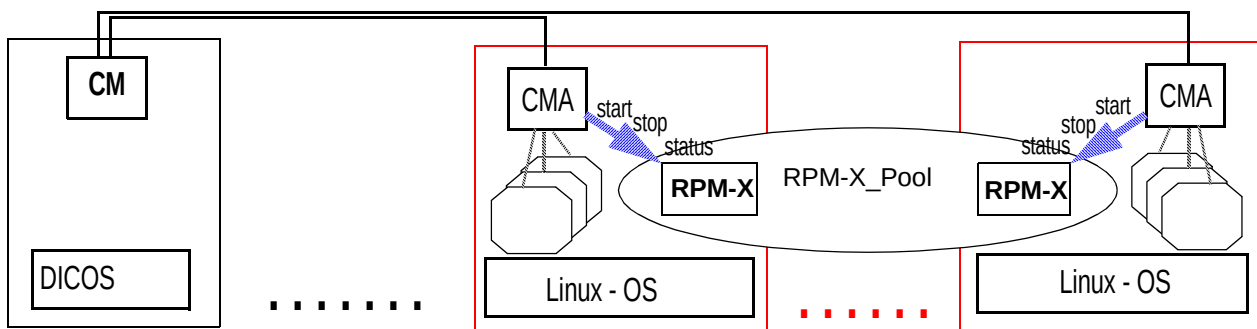
Linux In The Cluster

Introducing Linux in the cluster contributes to the openness of the TSP system. Another advantage is faster development by using already available Linux SW. Linux is used in three different ways in the cluster:

1. As an operating system for the IO-processors
2. To execute “TelORB processes” on the linux processors
3. To execute “ordinary” Linux SW, packaged as RPMs (Redhat Package Manager)

The Linux processors employ a 2.4 kernel and some contents from a RedHat 7.1 distribution that has been stripped down for security and memory footprint reasons.

CM: Configuration Manager
CMA: Configuration Manager Agent



“TelORB Processes” on Linux Traffic Processors

Everything executing on a traffic processor executes in processes. A process is a separate execution environment. It executes in parallel

(pseudo-parallel) with other processes. The execution in one process cannot effect the execution in another except through special communication mechanisms.

‘TelORB processes’ can execute on Linux. A “TelORB process” on Linux, is basically a Linux process with access to the TelORB specific process environment.

However the following details should be noted when executing TelORB processes on Linux :

1. Starting a process on Linux involves higher overheads than starting it on TelORB.
2. The database is distributed on the TelORB processors only. The database can be accessed from Linux processors via a database client (see fig below) but only from Java. This implies worse performance than accessing DBN from Dicos processors.

Beacause of these, it is recommended to execute time critical tasks on TelORB processors, especially if DBN access is involved, since you cannot achieve any co-location between processes and data on Linux.

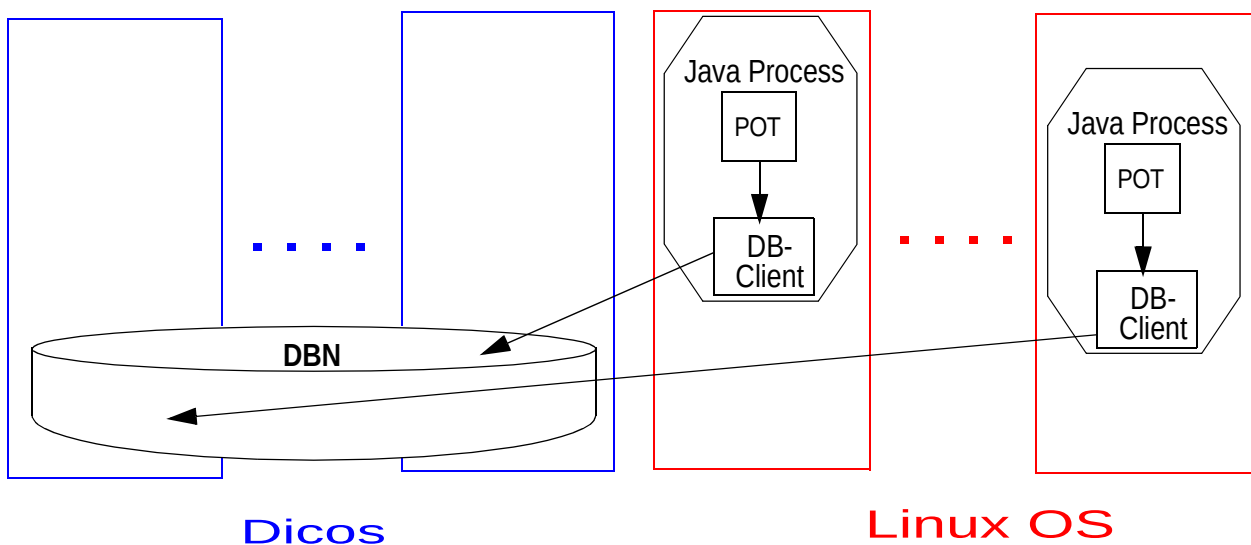


Figure 6. Linux Processors accessing the DBN

RPMs

You can also package “normal” Linux software as a so called RPM (Redhat Package Manager) and install it in the cluster. This allows you to bring existing implementations (freeware, shareware, 3rd party) straight into the cluster with very little effort.

TelORB

TelORB is a distributed real time processing environment and is the actual heart of TSP, especially when it comes to traffic and subscriber data management. It proves a robust platform for telecommunications applications. Some of the key features of TelORB are:

- A distributed object oriented operating system
- Provides an object request broker (ORB) according to Corba 2.0
- Provides a Java virtual machine,
- Development environment on Sun/Solaris or PC/Linux
- Database services
- Upgrade ability

TelORB is a distributed operating system with real-time characteristics suitable for controlling telecom applications. It can be loaded onto a group of processors. The group of processors will behave like one single system. Applications can run on top of TelORB. Different parts of the application can run on different processors. One part can communicate with another in a transparent manner. If there is a need to increase the processing capacity you can add more processors in run-time without disturbing ongoing activities. The increase of the capacity is a linear function of the number of processors, therefore TelORB is described as a truly scalable system.

TelORB Characteristics

- Distributed
- Object Oriented
- Fault Tolerant
- Open
- Scalable, High Performance
- Real-time characteristics

TelORB's Operating System is highly reliable and is designed to keep applications running whatever happens. Regardless of whether a processor goes down, an Ethernet Switch fails, the hard disk goes wrong, an application running on TelORB will still carry on doing whatever it was designed for.

Distributed

TelORB's Operating System is distributed. That means applications run in processes that do not exist on only one of the TelORB processors, rather they are distributed across a number of processors.

Object Oriented.

TelORB is not only designed for object-oriented programs, the Operating System itself is object oriented.

Fault-tolerant

The high availability indicates that the system is designed in a way that it should never go down, either during the normal operation or when expanding system is under expansion.

A properly designed TelORB application is extremely fault tolerant. The fault tolerance is achieved primarily by duplication. Every piece of equipment that is vital to the operation of TelORB is duplicated. Programs and data are also duplicated, each piece of data that makes up the TelORB system, both the programs and the database, is stored on more than one processor. This solution does not apply just to the applications that run on TelORB, it applies to TelORB itself. This combination of duplicated hardware and distribution of software means **there is no single point of failure in a TelORB system**, it means that one fault alone (a processor crash) will not reduce the service availability of the system.

There is also an option to have Disk-Logging and Network Redundancy within the TelORB system.

Open

Openness means that the system is non-proprietary. TelORB is an open system in several ways.

- The hardware used in the TelORB system is all commercially available from many different suppliers.
- The programs that run on the TelORB system are written in C/C++ or Java.
- The external protocols of the system are TCP/IP and IIOP (Internet Inter-Orb Protocol). TelORB includes a so called ORB implementing the IIOP, hence the name TelORB. IIOP is a part of a standard called CORBA (Common Object Request Broker Architecture). If an ORB follows the architecture of CORBA then it can exchange information with other CORBA ORBs. Thus the system is open for communication with external systems.

Scalability

Scalability means that the TSP system capacity can be increased by the simple addition of processor boards. Traditional telecom systems normally require either the introduction of more powerful processors and corresponding system upgrade or system split which is highly expensive and complex procedure. With the TSP system the whole procedure of capacity expansion is simplified. Also scalability is linear, meaning that the addition of processor boards increases capacity linearly. This linear scalability, see figure 7, makes TelORB capable of handling large volumes of computing, like in large nodes requiring high performance. In addition, adding processor boards does not require any system interruption; processor boards can be added or removed without having to bring down the TelORB environment.

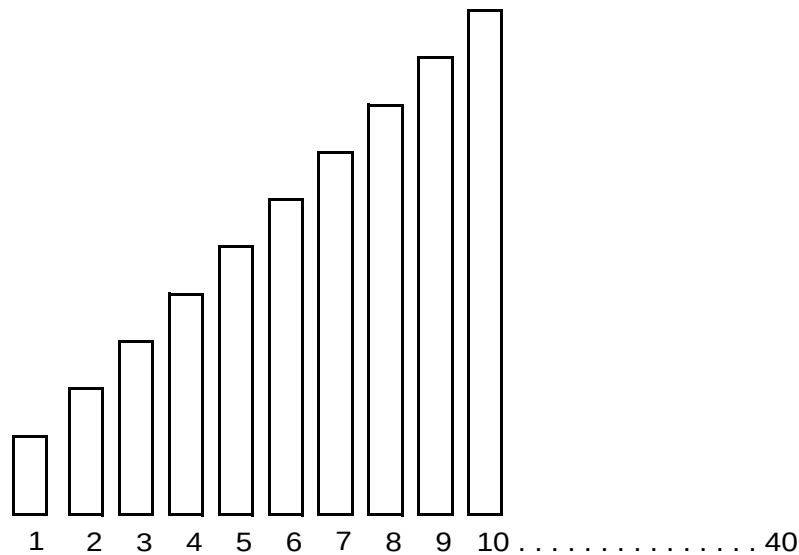


Figure 7. Capacity grows as processors are added

Realtime

The Real-Time response offered by TelORB is a "soft" real-time. Rather than the "hard" real-time, "soft" real-time is based on queuing theory.

More TSP Features

Interzone T-IPC

Interzone IPC allows one zone to send TIPC messages to another.

Application Log

The new application log feature allows an application to send log-messages to a file on the IO-machines.

File System

A new distributed filesystem is included in TSP 4.0. The file system supports NFS mounting to allow totally transparent access of remote disks. NFS mounting is used by the operating system itself, but can also be used by applications to access the file systems on the IO processors.

Corba

TSP supports the CORBA Policy mechanisms from CORBA 2.3.

A new feature in TSP 4.0 is to allow Secure CORBA interactions between external clients and servers and TelORB clients and servers.

Java

TSP supports the Java 2 Standard Edition, JDK version 1.3, on both operating systems.

Software Architecture

The figure below gives an overview of the software in a TSP system. As operating system, the TSP/TelORB specific OS Dicos is used, but there is also an option to use Linux for some of the processors in the cluster. When using Linux the API's are almost exactly the same. The main difference is that the database is not distributed across the Linux Processors.

There are a number of processors, 5 in this example, interconnected through ethernet switches (not represented in the picture). These processors are connected to two IO processors (Linux machines) involved in I/O operations like initial loading, backups, dumping and upgrades.

The kernel of the operating system is distributed in primary memory of each processor.

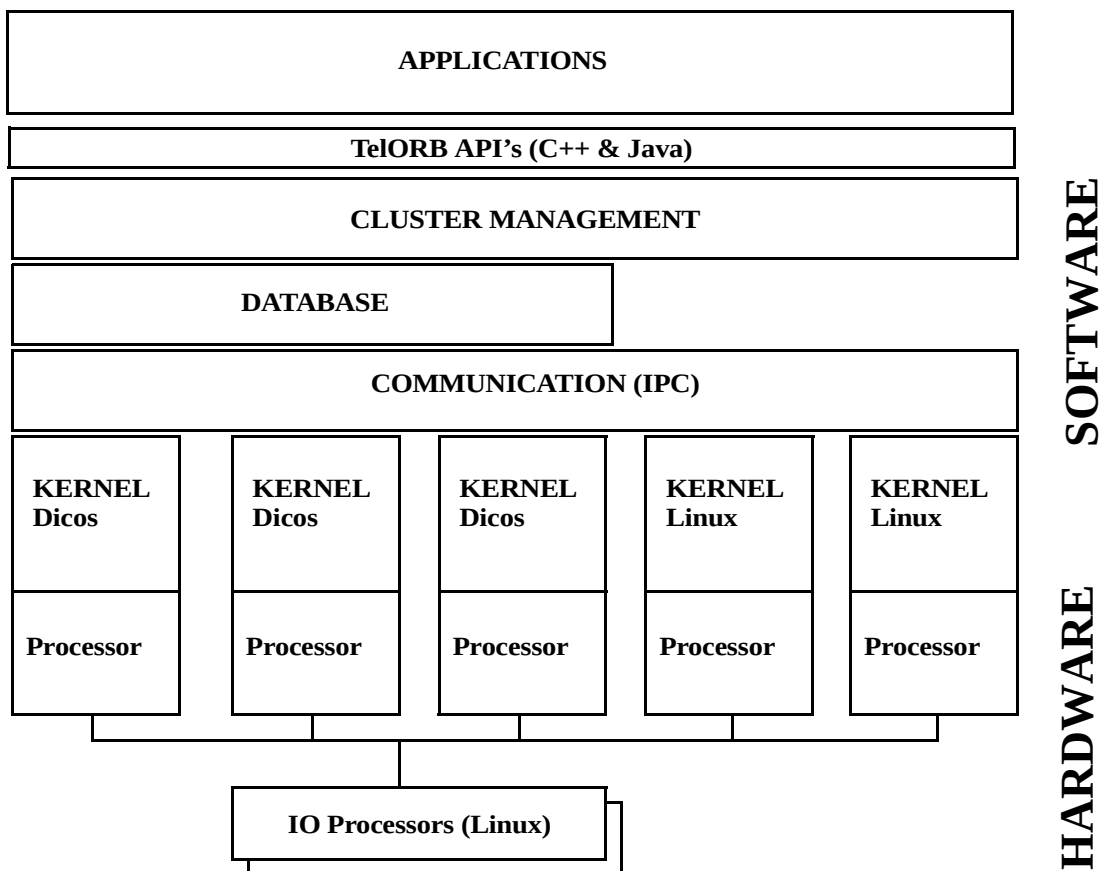


Figure 8. Software Architecture

The software is divided into the following layers:

KERNEL

The kernels that exist on each processor provide the processes, threads, memory management, scheduling and so on. They adapt the timers, interrupts and other facilities provided by the processors. TelORB's dis-

tributed object oriented operating system is known under name of Di-cos.

COMMUNICATION

The Communication Layer of TelORB connects the different processors to each other enabling inter-processor communication (IPC).

DATABASE

The Database layer provides persistent storage in primary memory that provides major performance advantages over disk-based storage. This means that the database is always held in primary memory, and distributed over the processors. Each individual piece of data is stored on more than one processor. In the event of a processor failure, the entire database is still available to the application.

SOFTWARE MANAGEMENT

The Software Management Layer automatically configures the executing software so that it runs efficiently on any processors that are available to the TelORB system.

CORBA-ORB

The CORBA ORB is included to allow the TelORB based system to communicate with other systems, including some of the graphical user interfaces that are used to manage the system.

JAVA VM

The Java HotSpot2.0 virtual machine is supported in TelORB. This makes it possible to execute TelORB applications implemented in Java. All processes implemented in Java that execute on the same processor share the same JVM.

HARDWARE

The TSP 4.0 hardware platform is based on commercially available hardware with high capacity and dependable node performance. Use of standard HW means low costs for installation, operation and maintenance. All processors are off-the-shelf, single board cPCI standard computers sourced from commercial suppliers. The use of industry standard hardware creates an architecture that is future proof, scaleable, flexible and cost efficient.

The hardware in TSP 4 is implemented using GEM hardware.

GEM

GEM, Generic Ericsson Magazine, is a generic platform component used in several products. GEM is a high-capacity, flexible and scalable magazine in which several functions can be combined.

The GEM concept can be used to build anything from an extremely small node to a very large one, using the same types of board and magazine. As capacity requirements grow, operators can add one or more GEMs with a suitable combination of devices, thereby smoothly extending the node without interruption of traffic.

Via the GEM backplane all plug-in units are provided with dual -48V power, duplicated 10/100 Ethernet and a maintenance bus. This causes the elimination of many cables, resulting in a very compact system.

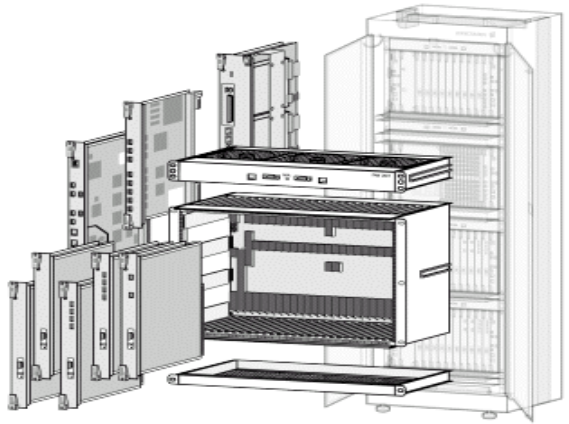


Figure 9. GEM Hardware

On the hardware level a TSP system consists of a cluster of Processor Modules (Processors). All processors are interconnected by duplicated internal Ethernet connections. In TSP 4.0 the hardware is essentially a “networked computer”, linking five types of Processing Modules:

- **PM** - Processor modules without disks and external interfaces
- **PM-E1/T1** - Processor modules with SS7 interface.
- **PM-ETH** - Processor modules with dual external Ethernet interfaces, used for VIP.
- **PMD-ETH** - Processor modules with disks and dual external Ethernet interfaces. Can be used as an IO.
- **PMDM-ETH** - Processor modules with disks, dual external Ethernet interfaces, DVD and tape streamer. IO 2.

The Processor Modules are all based on the same sourced industry standard cPCI board. The cPCI processor boards is a Teknor/Kontron MXS64GX with a Pentium III, 700 MHz processor. The processor has 256KB cache and 1GB SDRAM. The board has also a SCSI interface. A Processor may run either TelORB or Linux, depending on configuration. The hard disk equipped processor modules can run Linux only.

PM - (Traffic Processor)

The PM (Traffic Processor) is diskless and does not have any external interfaces. The traffic applications are executed on the PMs, running Dicos or Linux. Each node contains a variable number of PMs, depending on the processing capacity needed.

These PMs may be referred to as standard TPs.

PM-ETH - (TP-IP)

A PM-ETH is a PM(Traffic Processor) equipped with two additional 100Base-TX Ethernet interfaces at the front for external network connections. These interfaces may be used to configure VIP.

These PM-ETHs may be referred to as TP-IP.

PM-E1/T1 - (TP-SS7)

A PM-E1/T1 is a PM equipped with four E1, T1 or J1 physical links. These may be used to configure SS7.

These PM-ETHs may be referred to as TP-SS7.

PMD-ETH (IO 1)

The PMD-ETH consists of the processor board and two 18GB hard disks. Two processors are dedicated to O&M. They also act as a gateway between the external and the internal networks. The two hard-disk-equipped PMs (=PMD) dedicated to O&M run the Linux operating system.

This PMD-ETH may be referred to as IO 1 or Support Processor.

There are other Processor modules with disk and these may support applications in a server node or may instead be used as Traffic Processors.

PMDM-ETH (IO 2)

The PMDM-ETH is basically a PMD-ETH extended with a DVD reader and a tape streamer (M for media). The DVD reader is used for input purposes, e.g. initial loading. The tape streamer is used for backups. It has support for 20 GB uncompressed data.

This PMDM-ETH may be referred to as IO 2, or Support Processor, and runs Linux.

ETHERNET SWITCHES

The Processor Modules are interconnected by means of duplicated Ethernet based LANs. There are two physical levels of the Ethernet switches:

SCB-RP:

Level 1 - switches on the SCB-RP (Support and Connection Board) boards that interconnects all the processors within a rack. The SCB-RP is the interface board for power and Ethernet connections to the processors. Each GEM rack includes two SCB-RP boards for redundancy.

GESB

Level 2- The interconnection of GEM racks is performed by using the Gigabit Ethernet Switch Board, GESB.

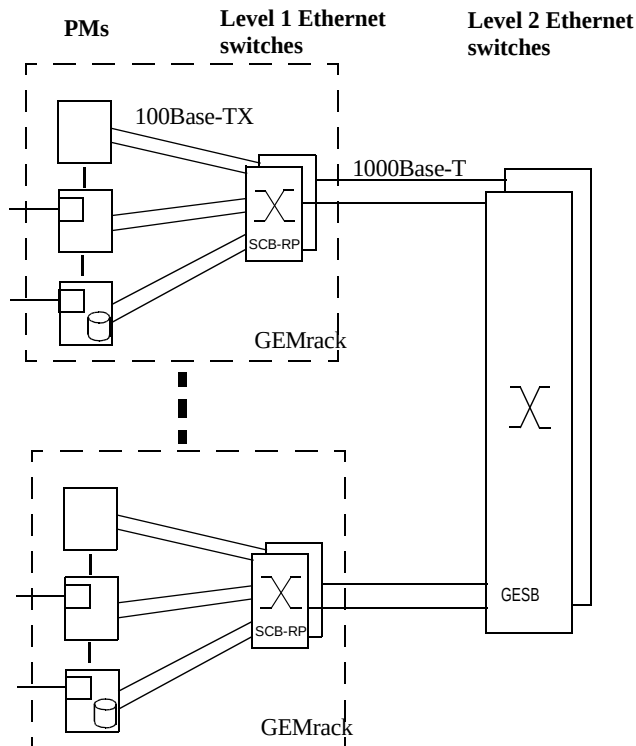


Figure 10. SCBs connect processors in a rack and then are connected to GESBs

PATCH PANEL

A patch panel is included in the cabinet. The patch panel consists of a number of Patch Panel Boards. There are two types of patch panel boards. One type is used for SS7 connections and the other for connection of Ethernet links, RS-232 and external alarms cables.

FANS

Above each magazine a fan is mounted. The fan unit consists of a low height subrack with 3 fans and a control board.

CABINET

The unit is housed in an Ericsson standard BYB 501 cabinet. Up to four GEM subracks can be housed in a BYB 501 cabinet. An expansion cabinet is available if additional processor modules are needed. All HW is accessible from the front panel, ensuring easy maintenance and replacement.

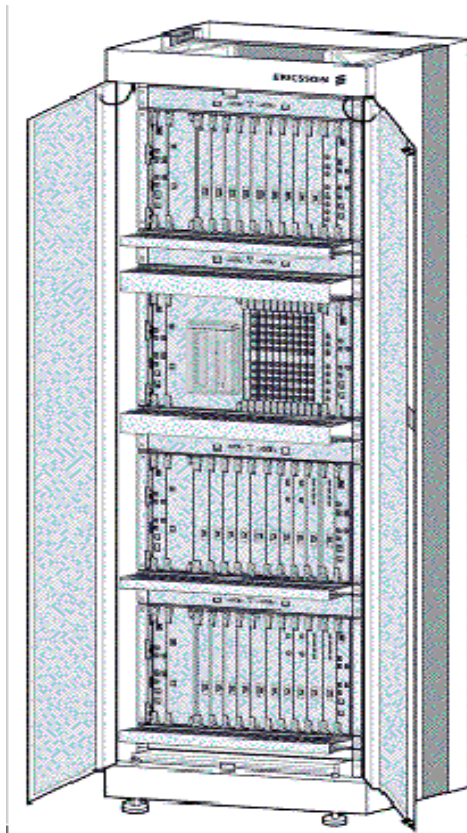


Figure 11. TSP Cabinet

Configurations

There are defined configurations for base and expansion cabinets. All cabinets are equipped with an infrastructure consisting of GEM magazines, SCB-RPs, GESBs, internal cables, including power cables. All expansion is made by adding PMs, PMDs and/or PM-E/T1s. If the sub-racks not are fully equipped, dummy units must be used for keeping the cooling air flow through the subracks.

Four magazines can be mounted in a cabinet.

There are four TSP 4.0 cabinet configurations:

- micro
- mini
- midi
- maxi

Micro configuration

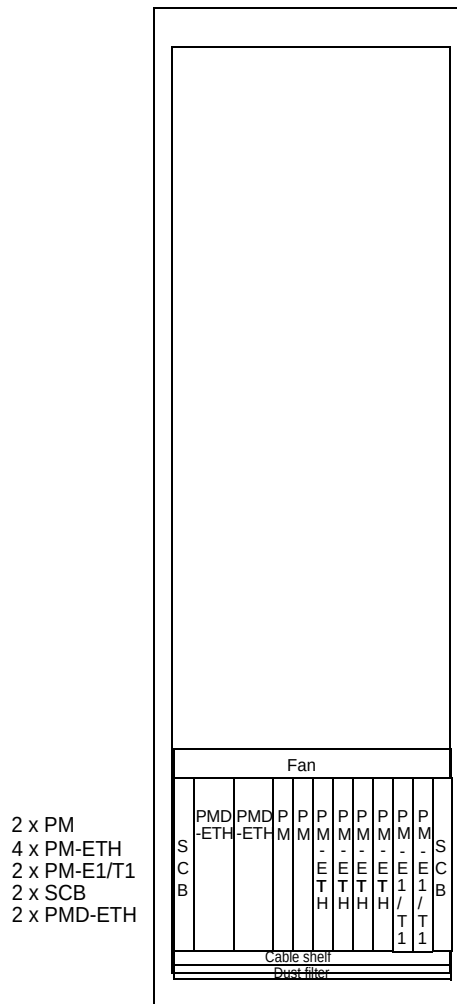


Figure 12. Micro configuration cabinet with 10 Pentium processors

Mini configuration

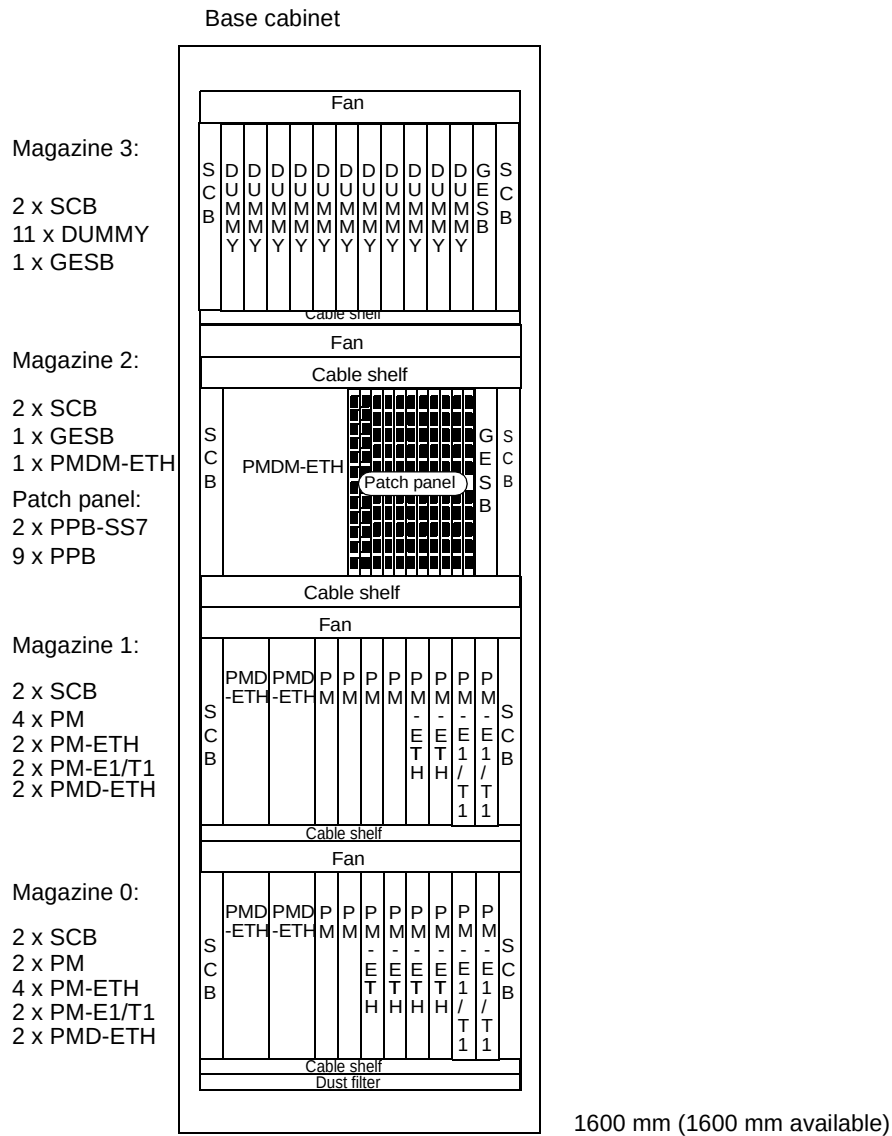


Figure 13. Mini configuration cabinet with 21 Pentium processors

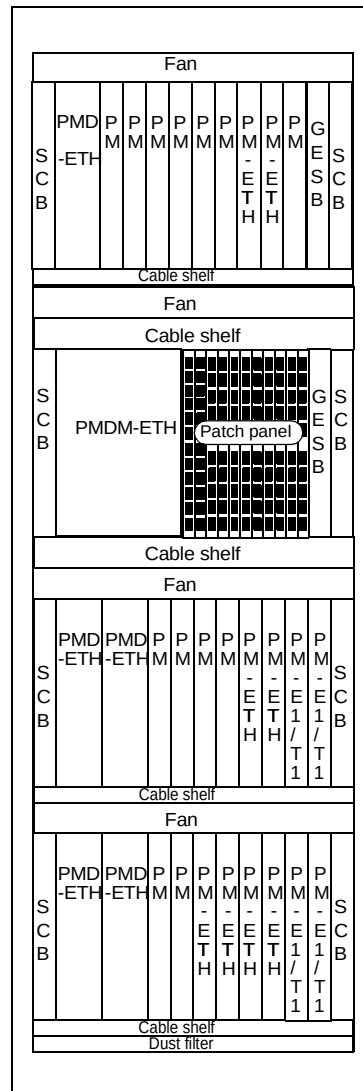
Midi configuration

Magazine 3:
2 x SCB
7 x PM
2 x PM-ETH
1 x GESB
1 x PMD-ETH

Magazine 2:
2 x SCB
1 x GESB
1 x PMDM-ETH
Patch panel:
2 x PPB-SS7
9 x PPB

Magazine 1:
2 x SCB
4 x PM
2 x PM-ETH
2 x PM-E1/T1
2 x PMD-ETH

Magazine 0:
2 x SCB
2 x PM
4 x PM-ETH
2 x PM-E1/T1
2 x PMD-ETH



1600 mm (1600 mm available)

Figure 14. Midi configuration cabinet with 31 Pentium processors

Maxi configuration

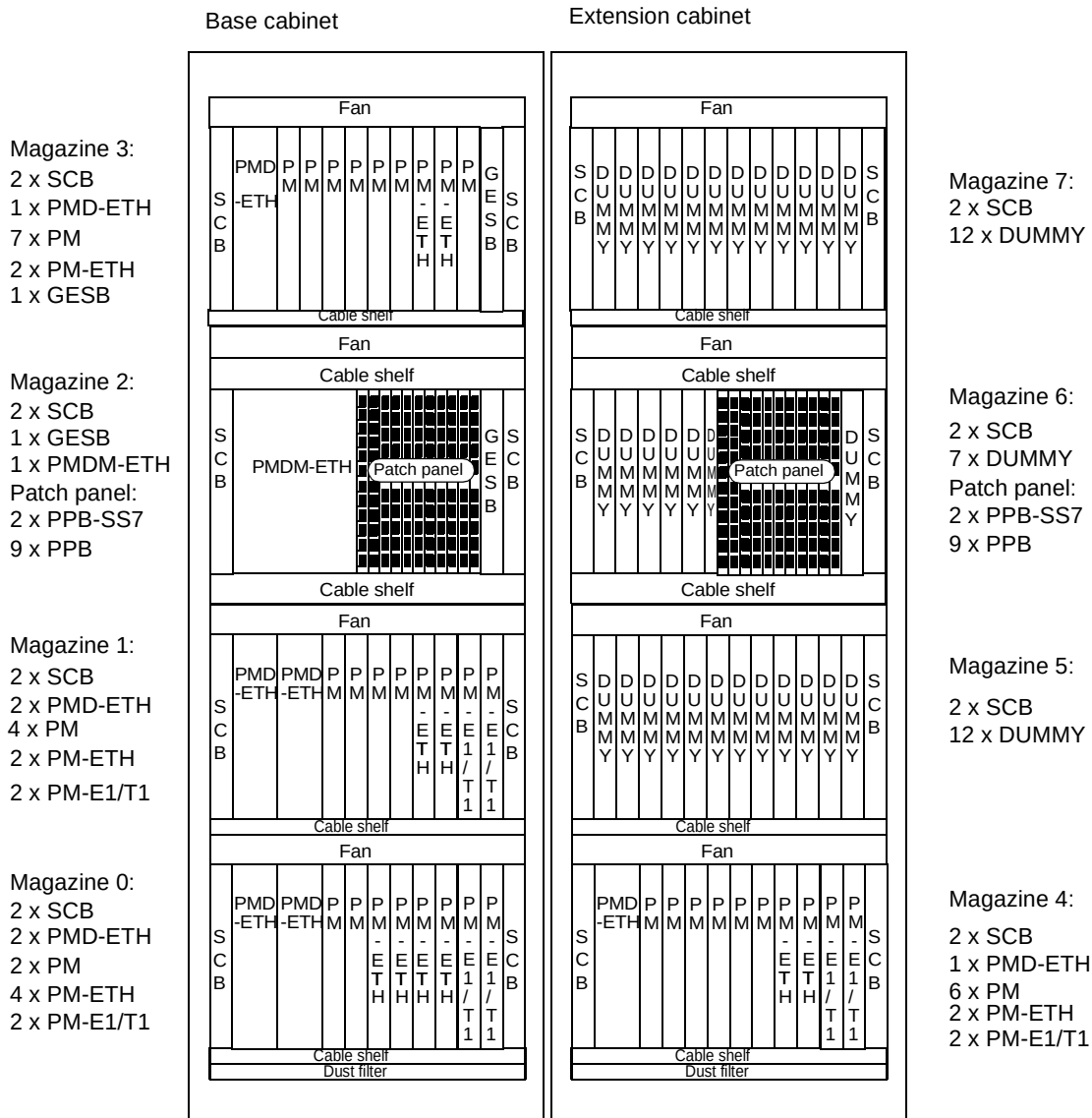


Figure 15. Maxi configuration cabinet with 42 Pentium processors

Processes

Processes are the fundamental executing objects. A process is an “independently executing unit”, it executes program code regardless of what other processes in the system might be doing at the same time. A process is specified in Delos. Delos is the TelORB specification language. A process is implemented in C++ or Java. The code for a process is found in one of the loadmodules loaded on the processor.

Process Interworking

Processes communicate by means of so called **Dialogues**. Dialogues are software entities specified in Delos used to handle communication between processes..

At a lower level Dialogues are based on IPC (Inter Process Communication). The IPC takes care of packing, sending, receiving and unpacking messages between processes.

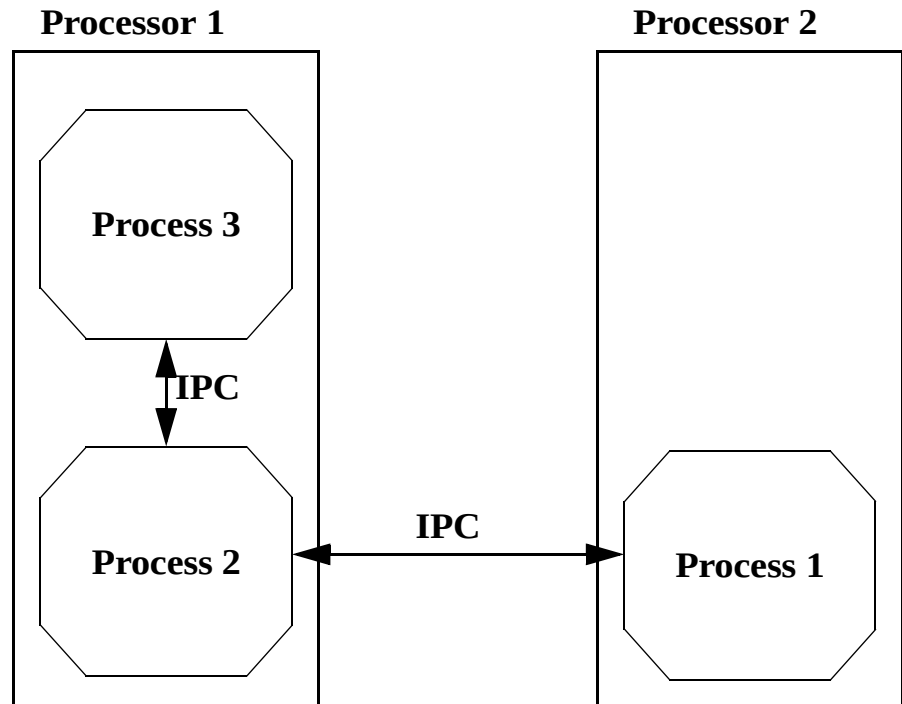


Figure 16. Inter Process Communication

Different types of objects

Within a process objects can be created and deleted. The most common types of objects are included in figure 17 below.

- Dialogue objects:
A dialogue object communicates with another dialogue object belonging to the same dialogue but in another process. The two objects are peers (as opposed to clients and server). Their communication is asynchronous (non-blocking). Dialogue objects are specified in Delos and can be implemented in Java or C++.
- Database Object - POTs (Persistent object types):
The POT is composed of several attributes (the actual data stored) and can have methods that implement operations on the attributes. Persistent objects are capable of creating, deleting, reading and modifying instances in the TelORB database. They are specified in Delos and implemented in C++. They can be used by Java code.
- CORBA objects:
CORBA objects are primarily intended for presenting interfaces to

functions implemented outside of the application framework. A CORBA server object can interoperate with a corresponding client object over an IP network. It is specified in IDL and implemented in Java or C++.

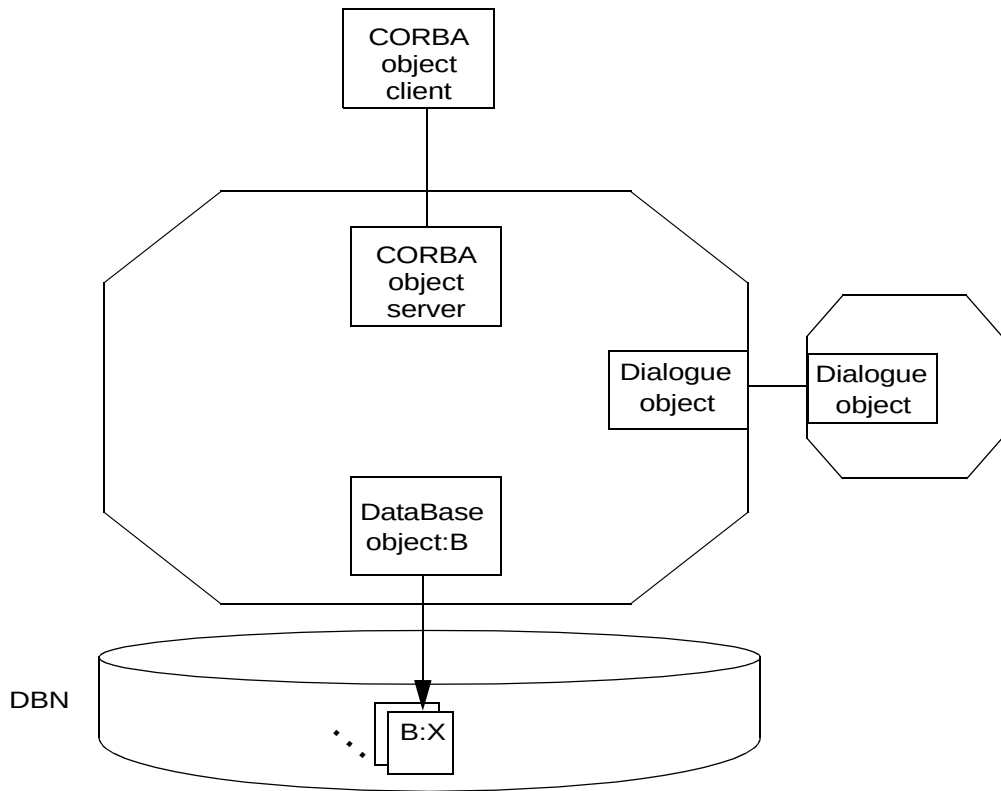


Figure 17. Different types of objects executing in a process

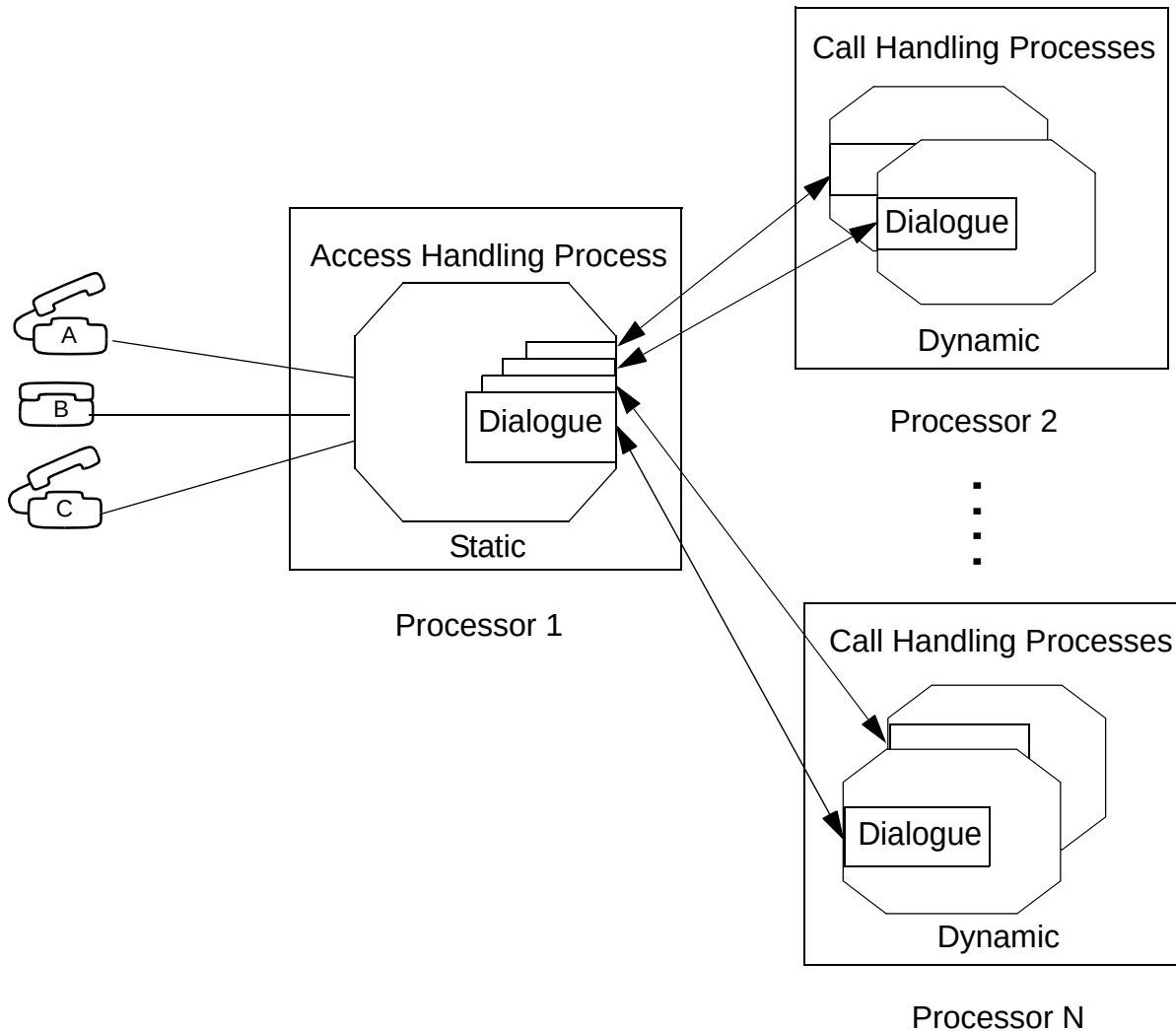
Static and Dynamic processes.

A running process is an instance of a Process Type. There are several varieties of process types, with different behaviour. The two general processes types are *static* and *dynamic*.

- Static processes are always running.
- Dynamic processes are started on request and terminated when their task is completed.

Using dynamic processes increases the robustness of the system. For example a static process could be supervising a group of subscribers (see figure below). When any of the subscribers makes a call, the static process will start up a dialogue and call for creation of a dynamic process handling the call for this particular subscriber. The static process could have a large number of dialogues with different dynamic processes. When the call is completed, the dialogue is closed and the dynamic process is terminated. The static process doesn't have to know where

the dynamic process shall be started, only the name of the process to be started.



If a fault occurs in one of the calls then only the corresponding dynamic process will be shut down and the other calls won't be affected.

Keyed Dynamic Processes colocated with data

A new instance of a keyed dynamic process is created whenever there is a call for it. It is a keyed process because the process name includes the primary key of a database instance. The process instance will be started on the processor holding the primary replica of the keyed database object. The advantage is faster access of the database since the process and the data is co-located.

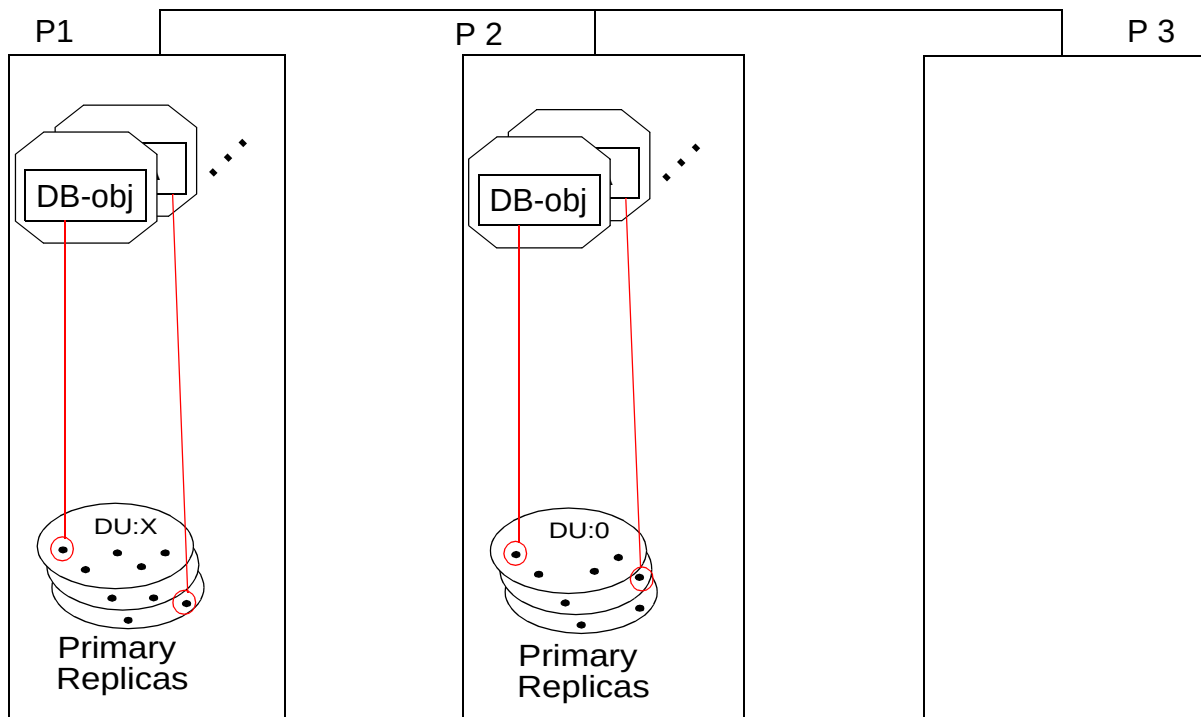


Figure 18. A keyed dynamic process type

Distribution controlled by CM

The TelORB distribution concept allows you to distribute two type of items:

- Process Types
- Persistent Object Types (POTs)

The distribution of processes and database data over the cluster, is controlled by the Configuration Manager (CM) executing on processor 1 or 2 (see fig below). The CM is the most central part of the software management layer. All processors have CM-agents (CMA) through which the CM can control the processes and data of the individual processors. The Processes executing on the LINUX Traffic Processors are supervised by the Configuration Manager (CM) just like normal TelORB processes.

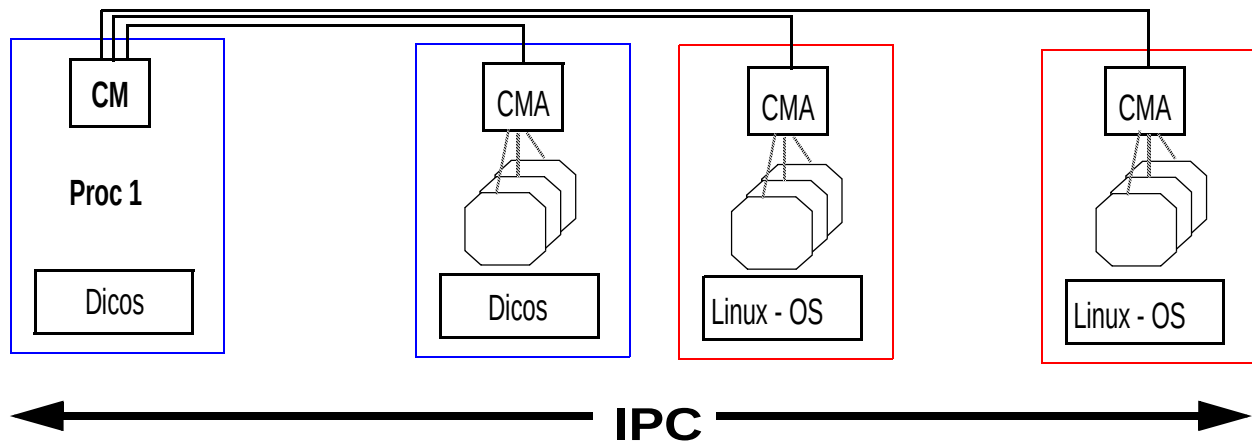


Figure 19. "TelORB processes on Linux"

Distribution Concepts

The fundamental distribution concepts are:

- Distribution Unit Type (DUT)
- Processor Pool
- Distribution Unit individual (DU)

Distribution Unit Types

The distribution unit type, DUT, is a container for process types and persistent object types. The distribution unit type is specified in Delos,. The DUT is later allocated to a 'Pool'.

Distribution Pools

All distribution unit types have to be allocated to a distribution pool. The DUT is allocated to a processor pool through a specification in a config file. The distribution pool is a virtual set of processors that restricts where database data should be stored and where processes are to execute. Physical processors are later associated with specific distribution pools in an 'epct' file.

DU individual

A distribution unit individual is a container for database instances (POT instances). We specify the maximum number of these containers - 'distribution limit'.

Distribution of processes

Exactly what processor a process executes on is generally determined by the process type's allocation to a distribution pool through its associated distribution unit type.

Distribution Unit Type and Processor Pools

The delos specification of a Process Type (PT) includes a reference to a distribution unit type (DUT) see figure 22 below. A process type is distributed as a DUT (Distribution Unit Type) and allocated to a certain Processor Pool, see figure 21.

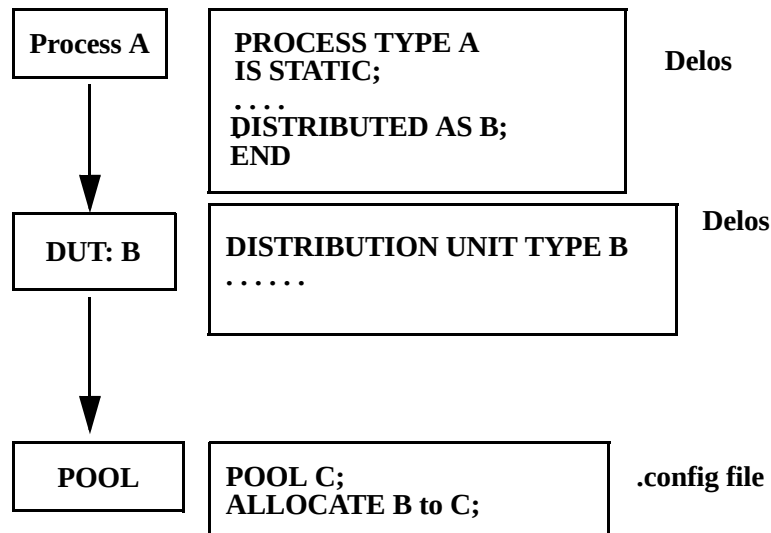


Figure 20. Process Distribution

A network engineer will prepare a site specific epct file. This file will, among other things, associate the physical processors to the logical processor pools (see figure 23 below). Please note that one processor can be a part of several processor pools. In the example below, Pool C has been allocated to run on Processors 1, 2 and 3.

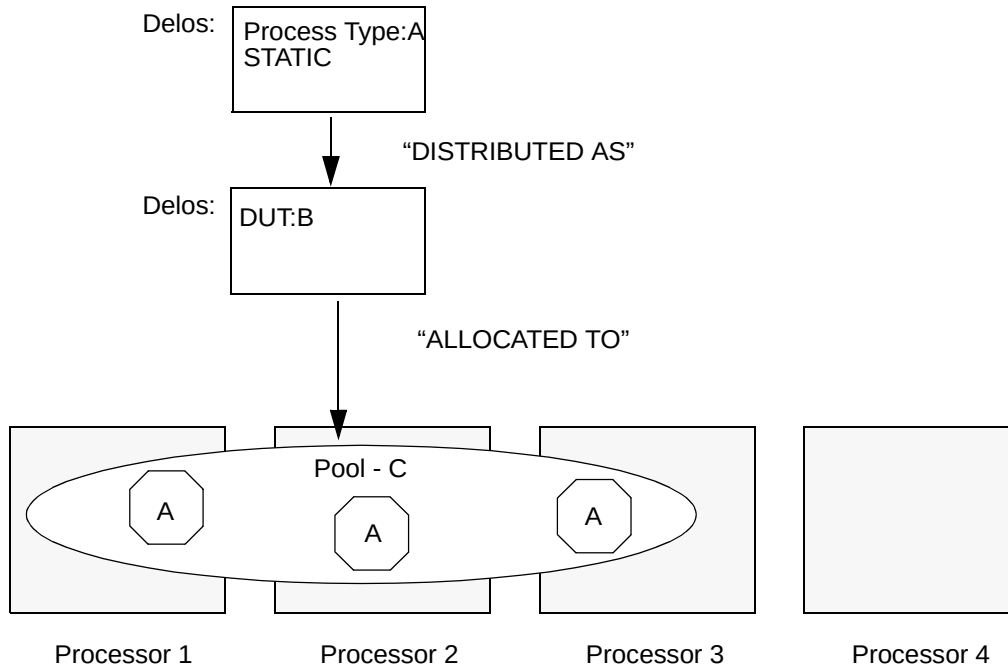


Figure 21. Distributing a Process Type

Distribution of database instances - POTs

Distribution Unit Type and Processor Pools

The delos specification of a persistent object type (database object) includes a reference to a distribution unit type (DUT) see figure 22 below. The DUT is specified in delos. It includes the distribution limit for the DUT. The distribution limit specifies the maximum number of DU (distribution unit) individuals. These DUs are containers for database instances. A high distribution limit implies more even distribution.

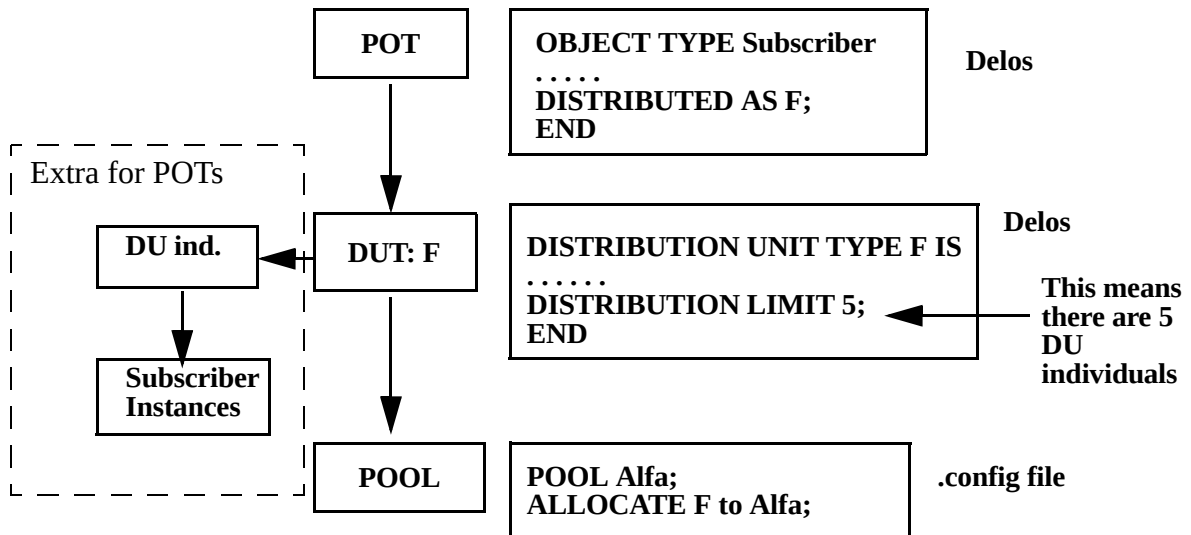


Figure 22. The designer specifies the distribution unit type for a database object or a process type. The DUT is allocated to a processor pool.

Processor Pools and Distribution Unit Individuals

Let's look at the DUT F alone. It is allocated to the pool 'Alfa'. In the 'epct' file this pool is configured to include processor 1 and 2. The distribution limit of the DUT is 5. It means there will be 5 so called distribution unit individuals. When the system is started the TelORB operating system will automatically allocate the 5 DU individuals as evenly as possible to the processors belonging to the pool. Three DU individuals will be on processor 1 and two on processor 2 (or vice versa).

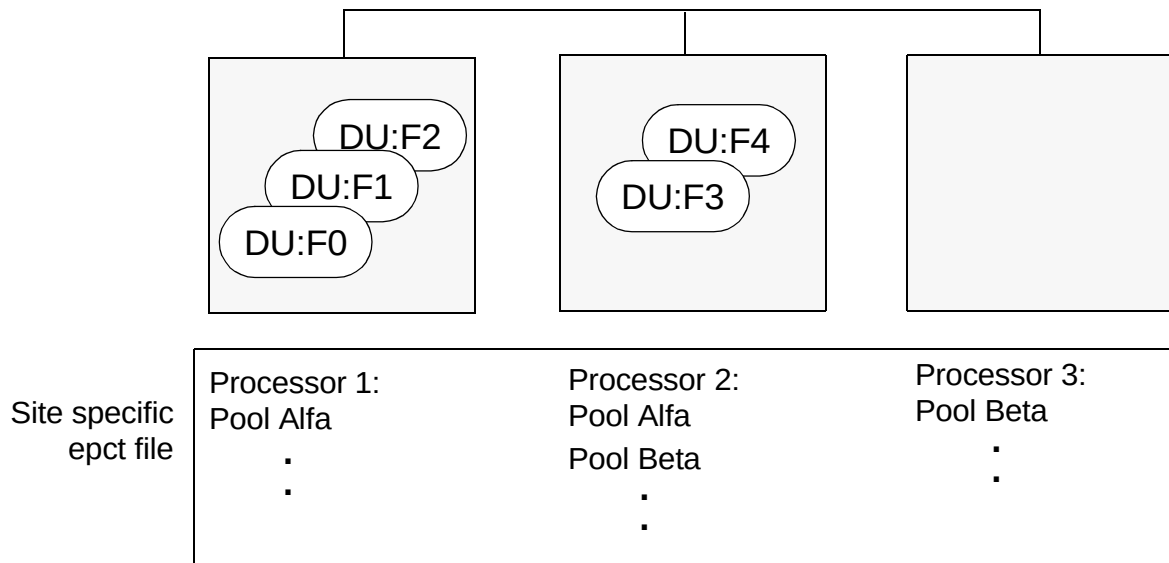


Figure 23. The processors of a particular site, are associated with the various pools in the site specific epct file.

Database Instances and DUs

A distribution unit individual is a container for database instances, we specify the maximum number of these containers - 'distribution limit'. In the following example the distribution limit is 12, i.e 12 DU individuals. The pool to which the DUT is allocated is distributed over 3 processors, according to the site specific epct file.

TelORB will automatically allocate the 12 DU individuals to the 3 processors, that means 4 DUs for each processor, see figure 25

Processor 1	Processor 2	Processor 3
4 - 5 10 - 11	0 - 1 6 - 7	2 - 3 8 - 9

Figure 24. How the 12 DU's are distributed among the 3 processors.

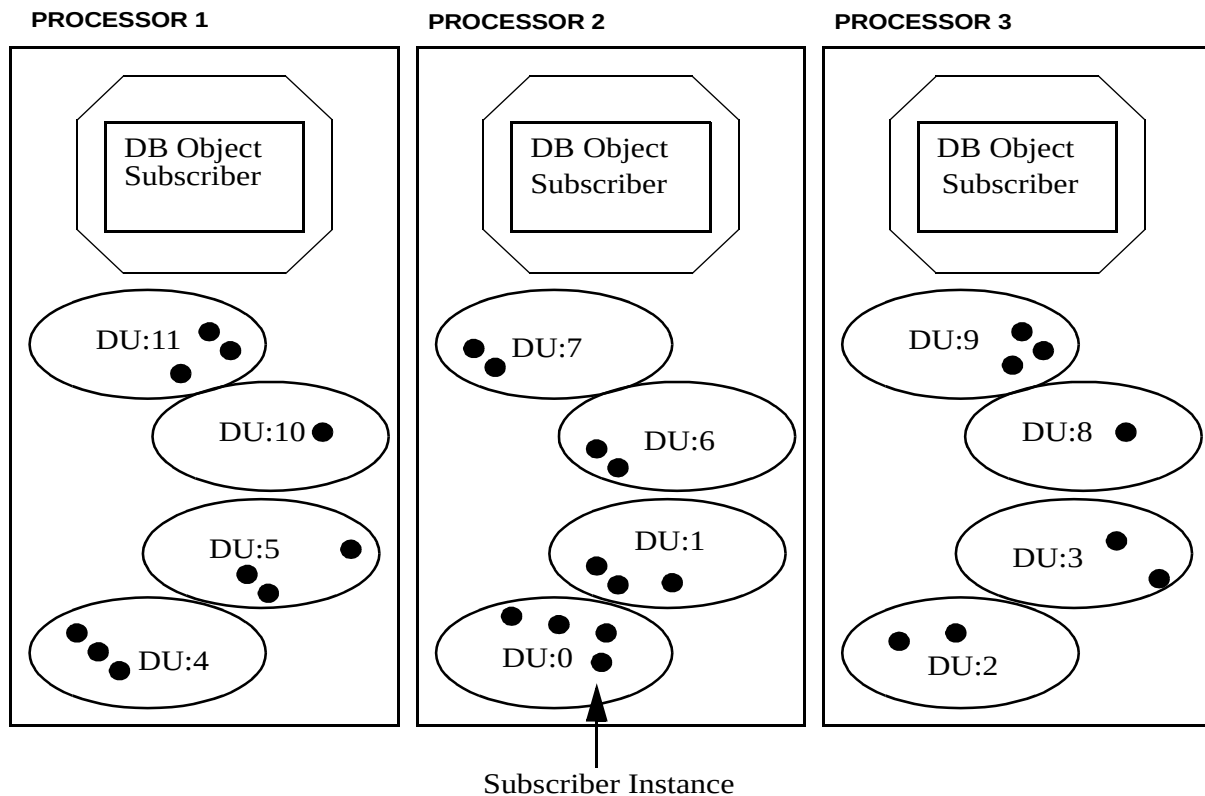


Figure 25. A DUT with distribution limit 12 allocated to a pool of 3 processors

Data Duplication

To achieve a fault tolerant database each piece of data is stored in more than one processor. This means that each DU individual exists in two copies, one primary replica and one redundancy replica. What has been described earlier see figure 25, was the primary replica of the DU individual. Considering the redundancy replica, the previous table, see figure 24, becomes the new one.

	Processor 1	Processor 2	Processor 3
Primary Replicas of DU's	4 - 5 10 - 11	0 - 1 6 - 7	2 - 3 8 - 9
Redundancy Replicas of DU's	2 - 3 6 - 7	4 - 5 8 - 9	0 - 1 10 - 11

Figure 26. Primary and Redundancy Replicas of DU's

Using this distribution it's clear that even if a processor crashes we have all the DU's stored in the remaining two processors.

Communication

Apart from the IO processors, external communication is also provided by the following physical interfaces:

- PM-ETH Processor Module (TP-IP) - This terminates Ethernet connections for IP signalling. From 4 to 10 PM-ETH modules can be supported on the platform. Each TP-IP PMC can be configured for VIP to handle external IP communications.
- PM-E1/T1 Processor Module (TP-SS7) - This terminates Signalling System #7 (SS7) signalling connections. From 2 to 6 PM-E1/T1 modules can be supported on the platform. Each TP-SS7 has 4 E1/T1/J1 connectors, from which 32 signalling channels can be distributed.

VIP

Interfacing a TSP node to an external IP based traffical network is done with the Virtual IP (VIP) addressing function. VIP is a TSP concept and implementation, that offers a distributed IP stack within TSP nodes. VIP makes it possible to address TSP nodes in IP networks with a single, common IP address representing the actual traffic node.

Benefits of Using VIP

VIP provides the following key advantages:

- Highly fault tolerant.
- Scalable.
- Multiple IP addresses.
- Distributed IP.
- Multiple protocols - VIP can be used by applications using TCP, UDP and raw IP sockets.
- Transparent - Existing applications are left unaffected by introduction of VIP in a network (valid for both client and server applications).

VIP fulfils three important requirements:

- Scalable physical input to the cluster - The IP signalling input to the system can be scaled as needed. An increased traffic rate can be met by adding new hardware and changing the software configuration accordingly.
- Single IP address - Server applications on TelORB can publish "URLs" or IP endpoints at which a particular service is provided without having to make external clients aware of the internal cluster solution. That is, a client can use the same (virtual) IP-address to access the system independent of which processor that in the end actually serves the request.

- Application support - The VIP application layer will help application designers to build distributed and scalable TCP/UDP/IP applications.

The VIP solution is mainly a software solution. A standard router and some Ethernet connections are the only extra hardware required. The router will forward VIP-messages to any processor in the TelORB cluster that is configured to act as a VIP front-end. The message will then be distributed internally in the cluster using the highly optimized TelORB IPC protocol.

An external IP network is physically connected to a TSP node via an intermediary external gateway router. The gateway router is thereafter connected to the TSP node via communication enabled processors denoted PM-ETH boards (TP-IPs).

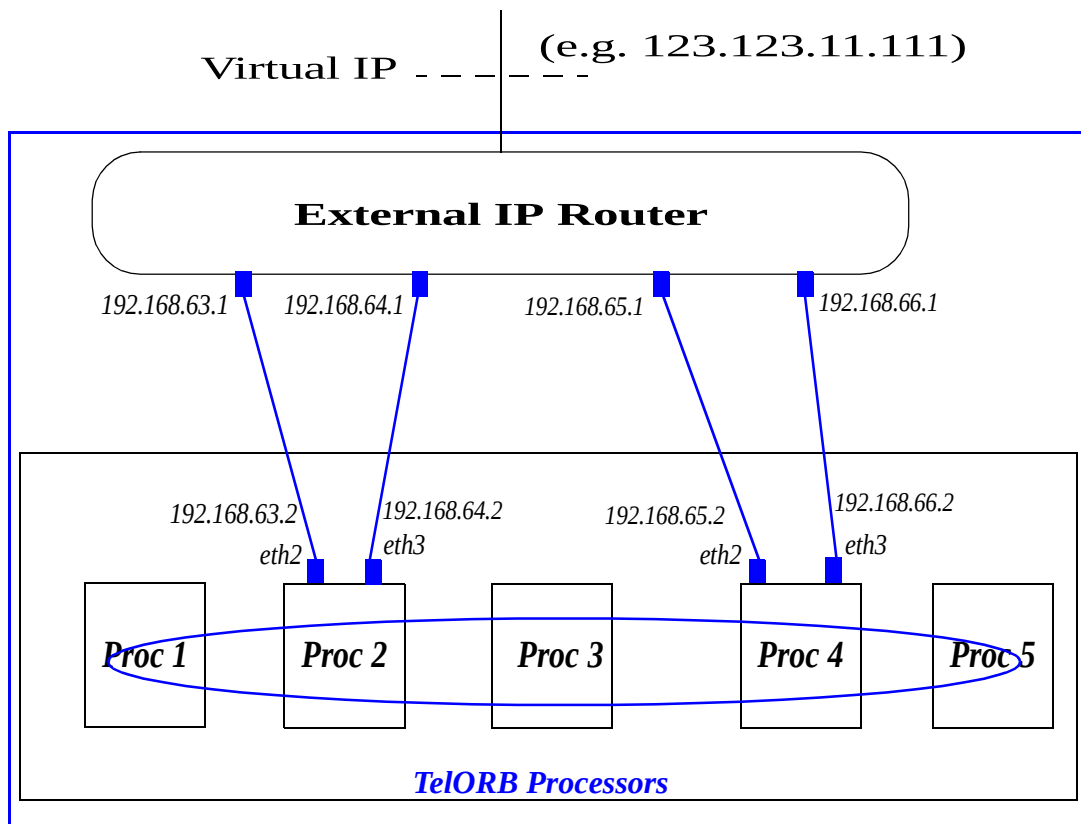


Figure 27. VIP view of a TSP system

VIP on LINUX

Full implementation of VIP together with Linux is planned. However, in the first TSP 4 releases the TP-IP boards interfacing the gateway routers will require TelORB/Dicos as OS.

To get more information on VIP and adding a VIP router link see:

SS7

The Signaling subsystem within TSP provides connectivity with external tele and data communication networks. SS7 protocol is still the most commonly used protocol for data interconnection in telecommunications networks. The TSP 4.0 SS7 Signaling stack is the latest in SS7 technology with a scalable implementation that can be distributed over the same Pentium processors that are used for traffic handling. The design includes a front-end process for termination of the E1/T1 interfaces and a back-end process for the upper layers of the stack. The modular design of the signaling stack allows signaling capacity to be increased in a cost-efficient manner since hardware is only added to the levels of the stack needing expansion.

Each PCI card configured as a SS7 processor has four E1/T1/J1s mounted on a Pentium processor which forms a front-end Signaling Terminal. Each TP-SS7 can handle up to 32 signaling links distributed over the 4 E1/T1/J1 connections.

CORBA

CORBA is a standard aiming at interoperability between different programming languages and platforms. It is a client server concept. For example a client implemented in C++ executing on a Unix platform is able to invoke operations on a server implemented in Java executing on Te-IORB.

CORBA Objects

CORBA client objects can invoke operations remotely on corresponding server objects. CORBA objects are specified in OMG-IDL (Object Management Group-Interface Definition Language). Based on one IDL specification, both client and server implementations of the CORBA object can be made. A server can be implemented in one programming language and the client in another. However there must be a so called ORB (Object Request Broker) implementation available in that language. Clients and servers can run on different computer platforms.

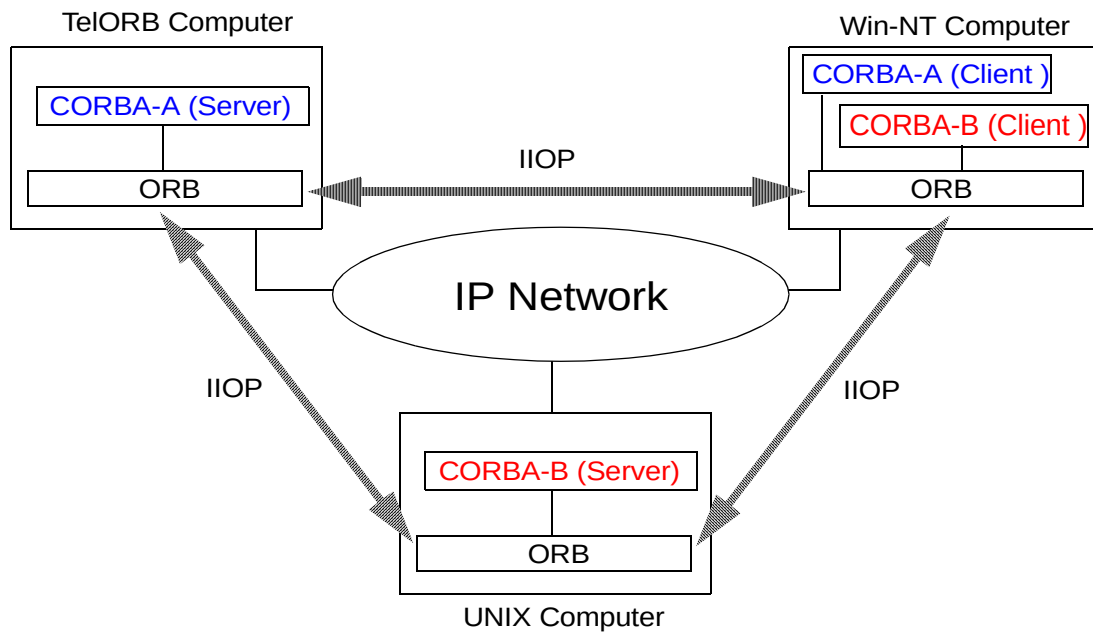


Figure 28. A CORBA client can invoke operations on a corresponding server object

An example of where CORBA is used in TSP is with an OAM GUI. In this case the GUI (TelORB Manager) is the client outside the TSP system. The user clicks the 'Backup' button. The implementation of a Backup is contained within the TSP system and this 'request' is sent via the ORBs and IIOP to the TSP system, where the Backup is implemented.

The Database (DBN)

A database object is an instance of a *persistent object type, POT*. The database is stored in primary memory of the processors.

DBN is used for storing information that requires persistency such as :

- long-lived data (longer than the processes using it)
- configuration data
- data that must be stored safely (e.g charging data)
- data that must be consistent
- data to be shared by several processes

The fault tolerant database included in TelORB is distributed.

The database objects are stored persistently, meaning that they survive different types of faults. This is implemented by having every object stored in at least two identical copies in write-protected primary memory on two different processors. In this way, the database is tolerant to:

- application software faults (that crash process instances),
- platform software faults (that crash one processor),
- hardware faults (that also crash one processor).

To further secure data, there is provision to backup database objects to disk.

Normal replication

The processors on which a database object is actually stored is determined by the allocation of the POT to a distribution pool through an associated distribution unit type. Currently storage of database objects is restricted to processors running TelORB.

Database objects can transparently be accessed from any processor in the zone

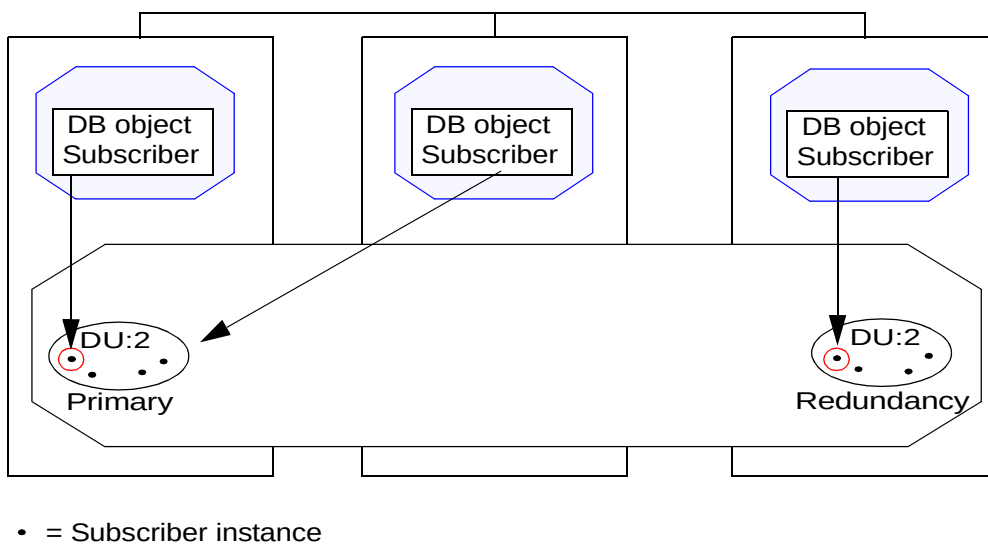


Figure 29. Database access with Normal Replication

One method already mentioned to secure data was to backup database objects to disk. There are other options to further secure data.

- Mirror database contents to another zone, a function called geographical network redundancy.
- Log updates to disk

Geographical Network Redundancy

The Geographical Network Redundancy function (NetRed) provides the means to configure two zones, located at some geographical distance from each other. Hot Standby redundancy is based on having these two nodes, where one is a primary and the other a stand-by node.

From a database point of view, one can say that a subset of the data is mirrored on another zone. Updates of objects are sent to another zone. This means that if ever the primary node suffers a complete shutdown due to circumstances such as total power failure, the standby node should takeover within approximately one second. When the broken node recovers, it becomes the stand-by for the other node, which is now primary.

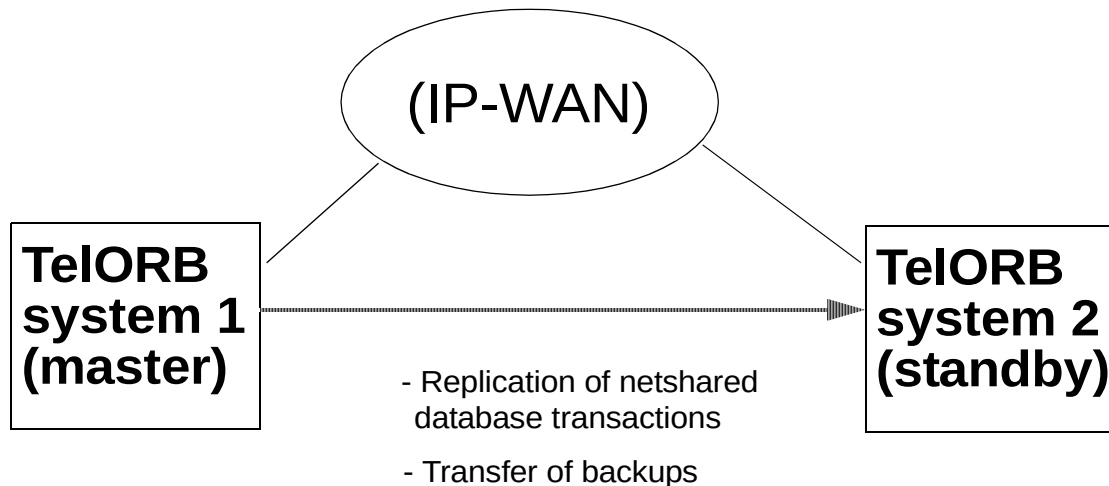


Figure 30. Hot-Standby Network Redundancy

Partitioned Netshared Redundancy

It is also possible to divide stored information into two partitions, where one cluster has the primary replicates of that partition and the other cluster acts as a standby to that partition. Vice versa is applicable for the other partition. Such a partition is said to be netshared.

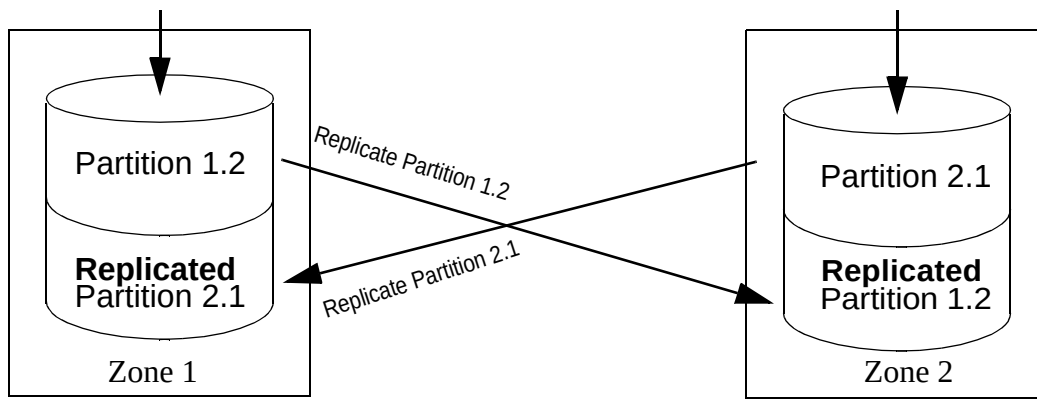


Figure 31. Partitioned Netshared Network Redundancy

Database Disc Log

Similar to network redundancy, persistent object types (POTs) can be configured (in runtime) to have disc persistence. A transaction that updates such data will store the changes in a log file on the IO processors when the transaction commits. If the zone should reload then the active backup is loaded (as always) and after that the database log files will be read to restore all disk persistent data.

This feature is only available for configurations that do not have the optional geographic redundancy option. Installations that do have geographic redundancy do not have the database disk logging option since their data is already being backed-up on another physical cluster.

High Availability Supported by TelORB Clusterware

High availability enables TelORB to handle software and hardware faults in a seamless way to the service. There are three types of faults. that may occur in the system

- Process fault
- Processor Fault
- Zone Reload

Different levels of recovery exist with increasing escalation factor.

Process Fault

If application software crashes, due to faults detected by the processor or faults detected by software, a dynamic process is simply terminated and a static process is restarted. The database is not changed at all.

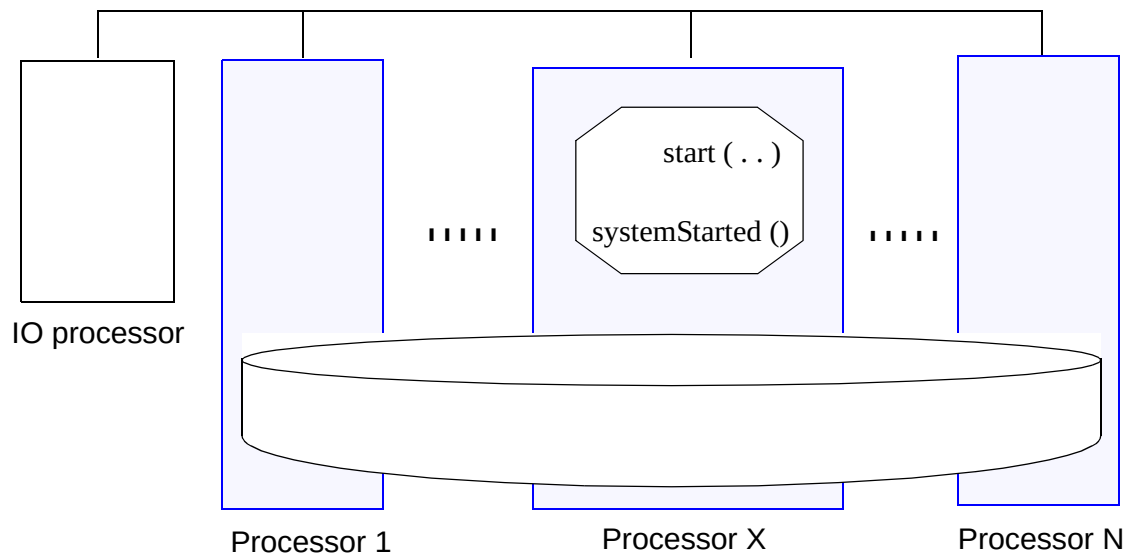


Figure 32. Process Reload

Processor fault

If a processor fails due to a hardware fault or some process deemed as critical, crashes, the processor where this happens is reloaded. The data stored in the DBN on this processor is instantaneously accessible on the processor where the replicated data resides while TelORB tries to recover from the fault.

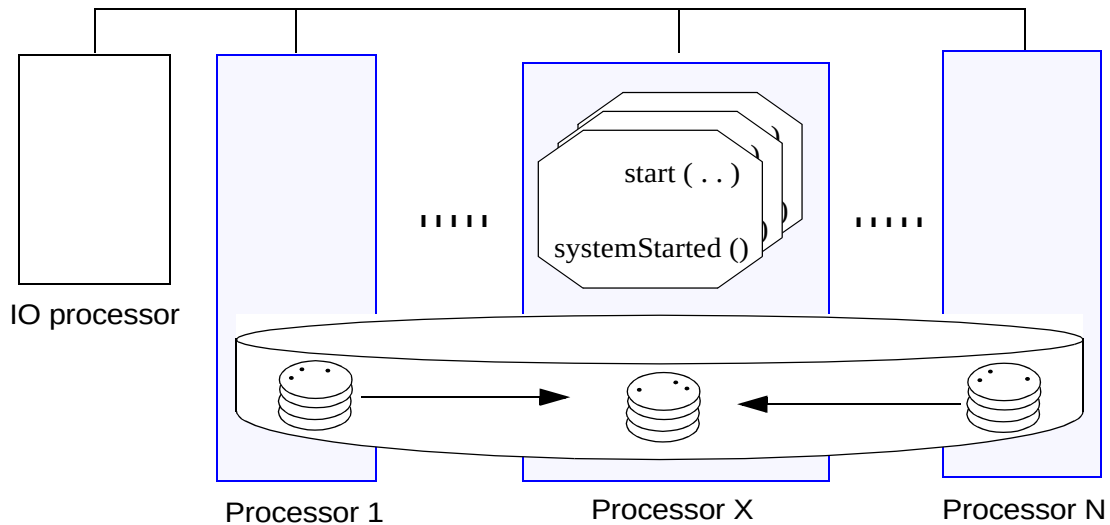


Figure 33. Processor Reload

Zone Reload

A zone reload is caused by having more than one processor unavailable at the same time. The configuration and the database is restored from a previously taken backup. There can be several backups on the IO processors but only one is “active”. A zone reload implies reverting to the state when the backup was taken.

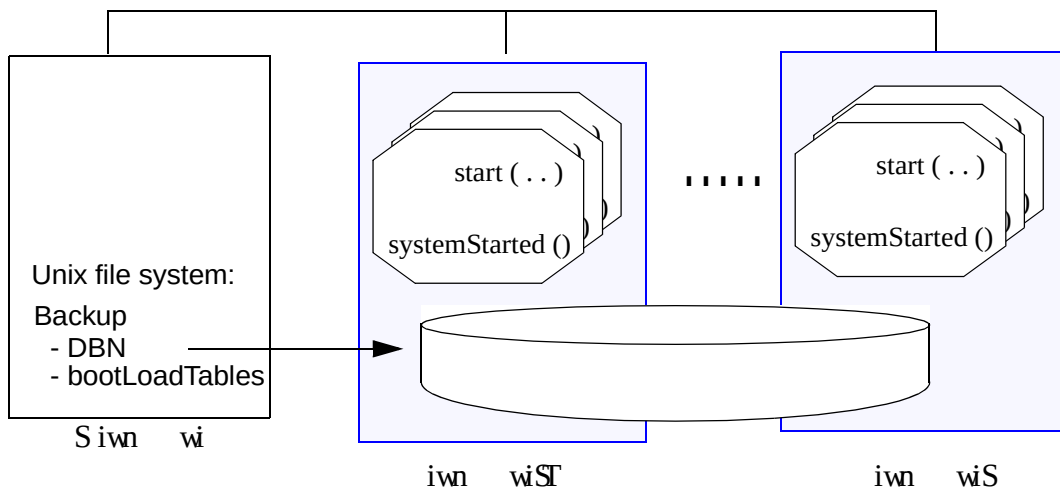


Figure 34. Zone Reload

Overload Protection

TSP4.0 has an overload protection mechanism that implements Call Shedding. When there are too many jobs in the job-queues, new incoming calls are shed. Conversely, when the number of jobs in the job queues declines below the threshold level, new calls are accepted.

Software Upgrade

Upgrade of both application and TelORB software can be done on the node with no disturbance to provided service. There are three methods of upgrading a system,

- The *smooth* method, used for most application software upgrades,
- The *redundancy* method, used for upgrading platform functions such as OS kernels etc
- *Upgrade by Reboot*, can also be used to upgrade TelORB.

Smooth Upgrade

With the smooth method, the new version of the software is loaded onto already running processors. Processes affecting ongoing traffic transfer their information from the old software version to the new software version and finally terminate the old software version.

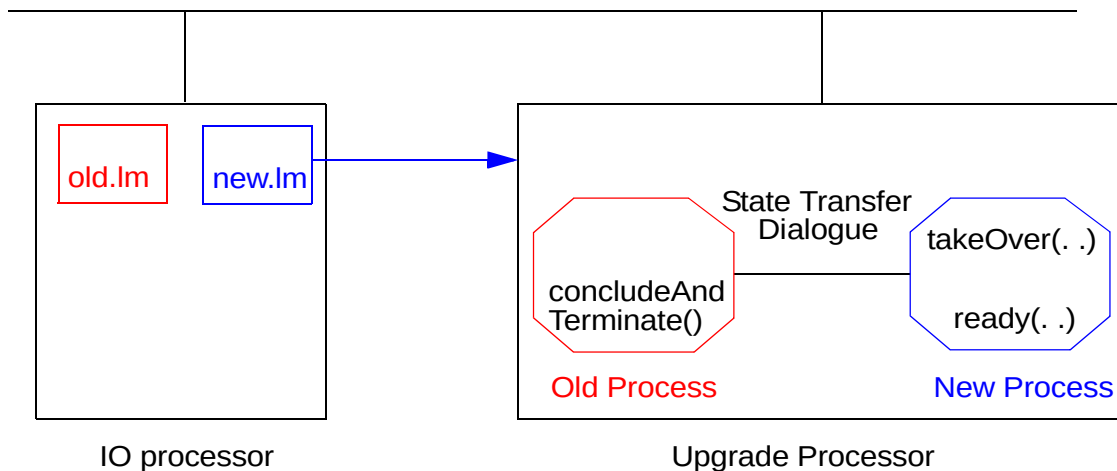


Figure 35. Smooth Software Upgrade

Redundancy Reboot Upgrade

With the redundancy method, the processors are upgraded one by one. This procedure begins by moving all ongoing traffic from one processor to other processors in the cluster without any disturbance to the service. The traffic load is evenly distributed among the remaining processors in the cluster. The emptied processor is upgraded with, e.g., the new version of TelORB, and the traffic is re-allocated to the upgraded processor. This procedure is then repeated for all processors in the cluster.

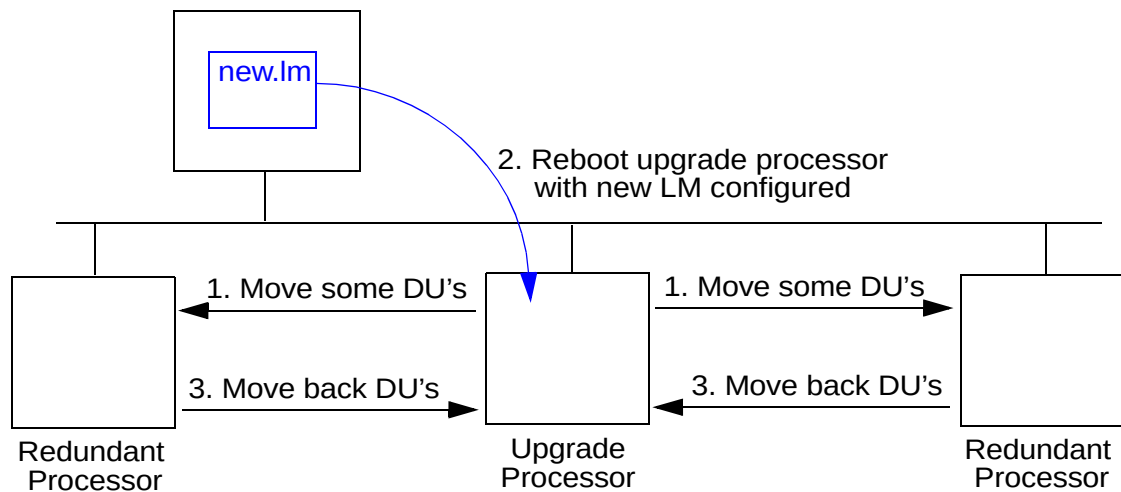


Figure 36. Redundancy Reboot Upgrade

Upgrade by Reboot

The Reboot method causes a reload of the entire TelORB zone, which means that the TelORB system is unavailable for the period while the reload is being performed. As a result, this method should only be used when the interruption will not affect traffic, i.e. when upgrading a system that has Network Redundancy.

Node Management

TSP Platform offers an Operation, Administration & Maintenance (OA&M) solution based on the Telecommunications Management Network (TMN) model. The Operation, Administration and Maintenance in the TSP node allows various network management operations, such as access to subscriber data, network data, access privileges, event recording, performance management and alarm handling.

Each TSP node incorporates a web-server which permits users to log onto the node using a standard web-browser. The web-browser is used to download a GUI client application, which then provides access to the management capabilities of TSP. No special-purpose local terminal is required. OAM can therefore be executed locally or remotely.

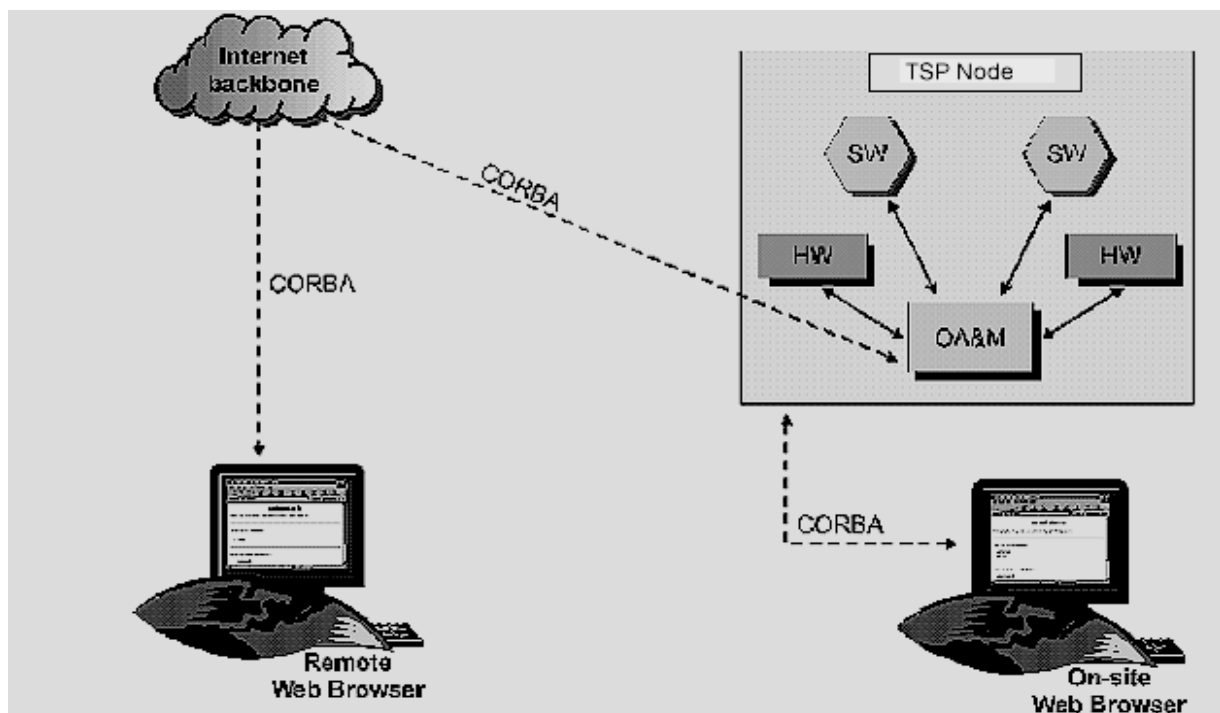


Figure 37. OAM can be executed locally or remotely

OAM Toolbox

TSP Platform offers all of the APIs needed to integrate the TSP node into a larger Network Management System (NMS). TSP also includes all of the necessary User Interfaces needed to manage the node without the need for an external NMS. These Element Manager GUIs are included in the TSP Node Management Toolbox which may be easily accessed from any standard web browser.

This toolbox is also called the Element Manager.

The Toolbox contains the following components:

- User Login GUI – for user authentication and authorisation purposes.
- Live Alarm Display (LAD) – for alarm presentation.
- Live Notification Display (LND) – for presentation of notifications.
- Ericsson Signalling Stack (EIN) GUI – for configuration of the SS7 stack
- JIM GUIs
- Query Tool – for querying logged information stored in the on-board relational database.
- LDAP Browser – for platform and application configuration.
- Performance Mgmt Dashboard – for on-line inspection of performance management data.
- TelORB Manager – for configuration of TelORB OS & CW. This includes task-oriented GUIs for:
 - SW upgrades
 - Backups
 - Processors
 - SW Inventory
 - HW Inventory
 - DNS/NFS configuration
 - Date & time management
 - ISP History
 - Network Redundancy
 - DBN Disk Log
 - Virtual IP configuration

TADE

TSP comes with a development environment and includes a software structure model, build support, and tools for configuring a plant. This development environment is known as TADE (TSP Application Development Environment). TADE can be used to efficiently develop and maintain applications running on TSP. TADE contains tools, documentation, design rules, training and support. TADE fully supports the Solaris and Linux platforms

Managed Items

The TSP design environment provides a number of command based tools, 'eptools'. The eptools are a vital part of the design environment for TSP. The eptools operate on a directory structure of Managed Items (MIs). Managed Items have attributes (for instance, an identity and version) and relationships to other MIs. A file structure is also associated with each MI type.

The following Managed Items are some of the more common MIs:

Software Interface (SWI)

Contains Delos and IDL, C++, and Java specifications of objects (i.e. APIs).

- Delos is a proprietary specification language, and provides the constructs that standard programming languages lack for specifying processes and database objects
- OMG IDL is a CORBA-OMG standard definition language

Object Unit (OU)

Contains C++ or Java implementation of one or several objects.

Load Module (LM)

Contains compiled and linked code.

Internal Delivery Package (IDP)

A container of LMs.

IIDP (Interface IDP)

A collection of SWI's can be delivered as an IIDP.

SCC (Software Code Component)

Contains implementations of ordinary C++ objects.

Eptools

The following are some of the eptools available:

epbuild

This tool effectively compiles (builds) and/or links a Managed Item.

epconfig

This tool manipulates your personal database. It allow you to refer directly to your MIs.

epgetpath

There is a database which uses the identity as the primary key and allows the path to be obtained from the MI's name. This is a lookup tool that searches this database.

epct

The configuration of a TSP system is performed by configuration files. This is the tool that configures the TSP system and application, it builds a runnable TSP loading group. With this tool you set up the relationship between the software and processing hardware.

eprun

This tool starts a Vega. Vega is is the simulated TSP Dicos Traffic Processor, that allows you to prototype applications without having access to a running TSP system

TADE Tools

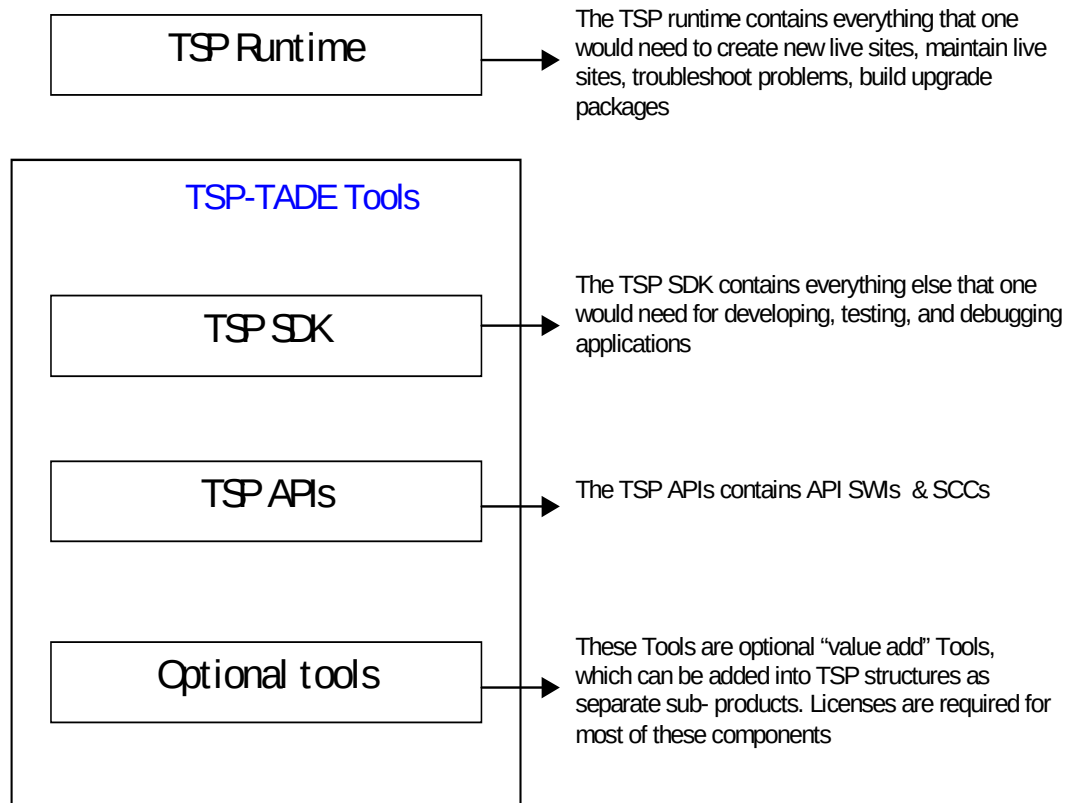


Figure 38. The TADE Tools overview.

Optional tools

These tools, unlike the SDK, are not mandatory. Components from the optional tools may be used and can be useful when designing applications on TSP. A separate license fee has to be paid to use most of the applications. They include:

- DPI (Development Product Information)
 - This aspect of TADE includes documentation of the tools and also how to design an application on TSP.
- Visual Modelling
 - Visual Modelling can be used with I-Logix Rhapsody or Rational Rose to make development of applications easier. The JIM MO Generator may also be used together with Visual Modelling.
- Dimensioning Tool
 - This tool is can be used to measure characteristics that are relevant to both the internal workings of TSP and the application.

From Design to Deployment

When designing applications on TSP, a specific design process has to be followed. The design process consists of

1. Implementation and Building
2. Configuration
3. Deployment is needed afterwards to start the application.

Implementation and Building

The building step consists of creating a Delos specification, which states the processes and POTs of the application. An IDL specification should be created also for CORBA objects.

When all these files are created, TelORB's tools for building (epbuild) can be executed. This tool starts all compilers to make IDPs (which include Load Modules). Delos specifications are compiled with the Delos compiler, 'delux'. The compiler generates C++ and Java code.

The epbuild command generates a number of C++ and Java skeletons. The desired functionality is obtained by adding implementation in these skeleton files. The designer will move these files to under the OU.

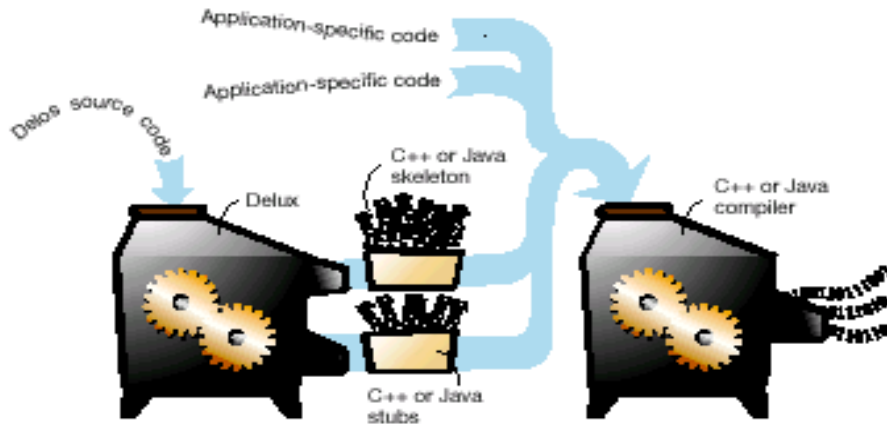


Figure 39. The Delos source code is fed into *delux* (the Delos compiler), which generates C++/Java stub and skeleton files. The application-specific code is then manually added to the skeleton files before final compilation (C++ or Java compiler) together with other application code. The IDL source code is handled in a similar way.

Configuration

The second step in the design process is configuration (with the *epct* tool). Using one or more configuration files (‘.epct’) the IDPs and LMs are configured for the application on the specified target system. During the configuration a selection of the required Load Modules will be made. The output file structure from the *epct* tool contains the LMs to be loaded, bootload tables and files that specify which managed object operations are needed to take the system into operation.

Deployment

To complete the process, the files must be deployed. Here the created files have to be copied to the I/O processor, from where the application can be loaded on a traffic processor via a loader. Then the target system will actually run the application.

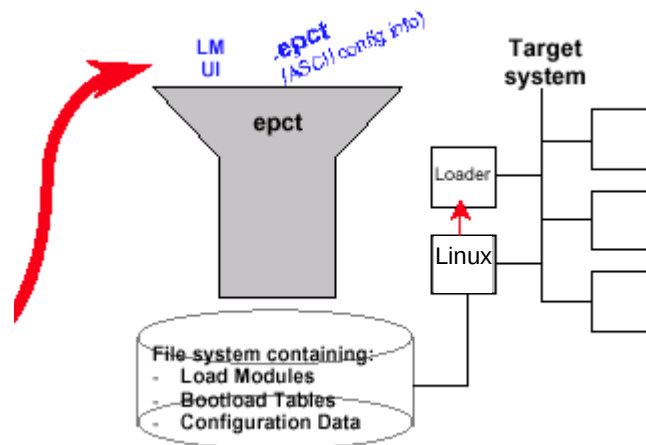


Figure 40. The *epct* tool generates a loadingGroup directory containing LMs etc which is then loaded onto the Linux-I/O Processors, thus allowing the system to load.

Configuring the Kickstart Server

A TSP cluster consists of two parts; boot-servers (IO) and clients (TP). The IOs are running Linux and contain all the software and configuration data, of which the Traffic Processors download a subset. When a TSP cluster is delivered from the factory, IO1 and IO2 are already installed. You do not need to use this procedure unless you need to reinstall IO1 and IO2.

An external boot-server (KickStart server) is used to perform an initial installation of an IO-machine in the TSP cluster. The kickstart server itself is not part of the cluster, it's temporarily connected to the cluster.

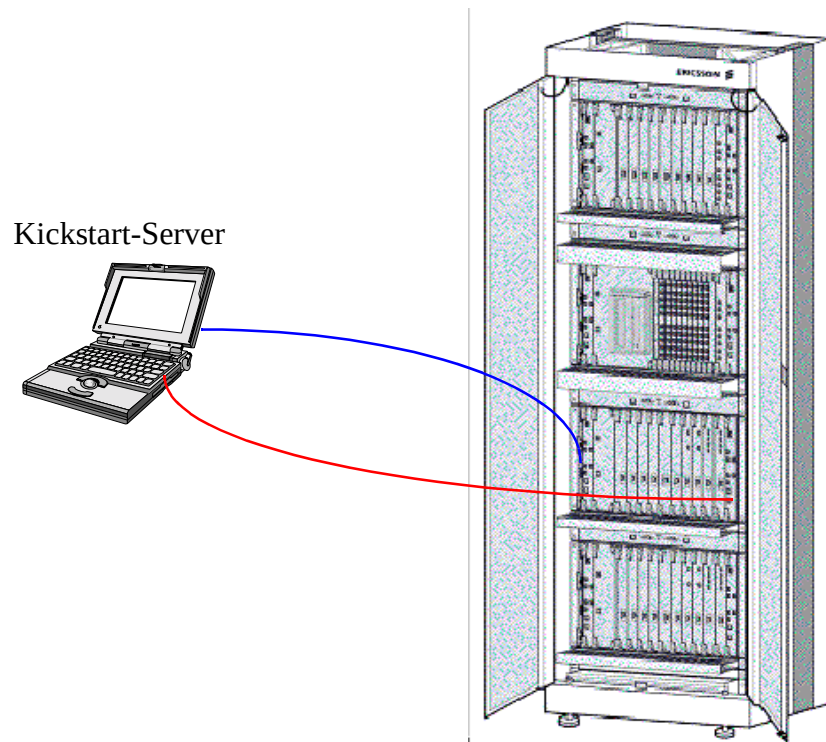


Figure 41. Kickstart server

The KickStart server is a standard PC, running RedHat 7.1 Linux, with some extra TSP-related software.

Kickstart Server Installation Instructions

The official documentation for these steps can be found at:

http://telorb.uab.ericsson.se/presentation/projects/TelORB/Doc/R14_Doc/DocStartR14withLinux.htm

Select the link “Installation of TelORB”.

There are 3 steps necessary to install and configure an external Kick-Start server:

1. Prepare the three necessary CDs
2. Install the software from the CDs on the kickstart server
3. Configure the ‘telorb.conf’ on the kickstart server

The kickstart server should be installed on a standard PC equipped with an Ethernet Network Card that’s supported by RedHat 7.1

1. Prepare the three necessary CDs

You need to burn:

- Two RedHat 7.1 ("Seawolf") CDs.
- Linux Kickstart Server installation set, Drop 2.3 on CD.

CD 1 - RedHat7.1: - ftp it from:

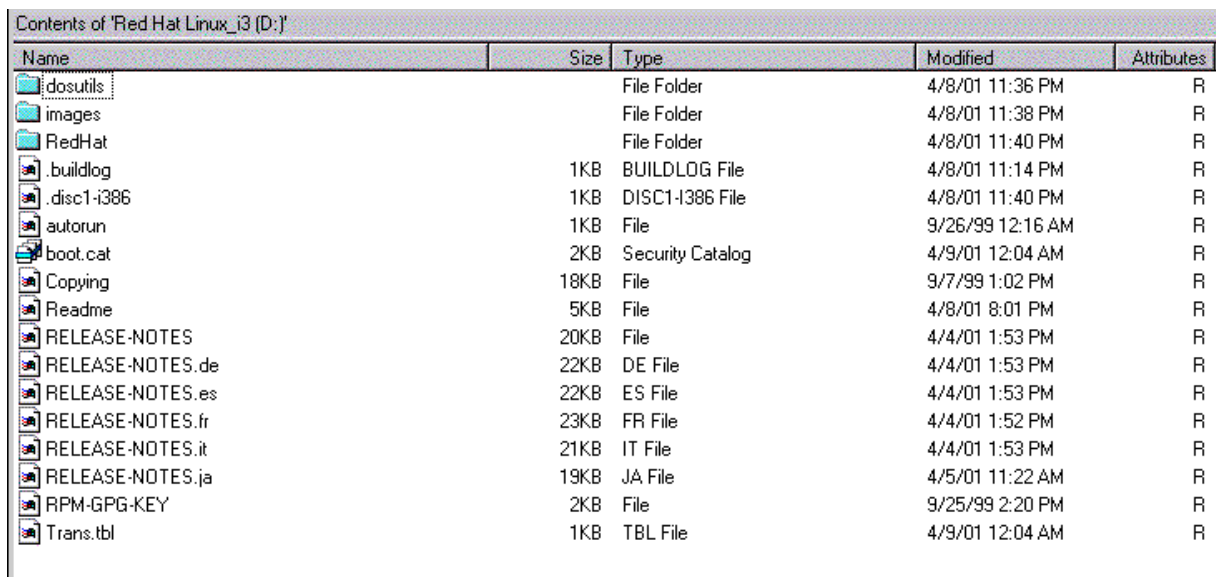
ftp://ftpmirror.ericsson.se/pub/MIRRORS/OS/Linux/redhat/redhat/linux/7.1/en/iso/i386/seawolf-i386-disc1.iso

CD 2 - RedHat7.1: - ftp it from:

ftp://ftpmirror.ericsson.se/pub/MIRRORS/OS/Linux/redhat/redhat/linux/7.1/en/iso/i386/seawolf-i386-disc2.iso

When you ftp the RedHat CD 1 you get a .iso file in your directory on the harddrive. To “unpack” this .iso file, you need a piece of SW called **isobuster**.

‘Extracting’ the .iso files will create a directory. “RedHatLinux_i3”, it’s the contents of this file that you should burn onto CD1 and 2.



The screenshot shows a file explorer window titled 'Contents of 'Red Hat Linux_i3 (D:)''. The window displays a list of files and folders with columns for Name, Size, Type, Modified, and Attributes. The files include folders like dosutils, images, and RedHat, as well as various files such as .buildlog, .disc1-i386, autorun, boot.cat, Copying, Readme, and several RELEASE-NOTES files in different languages. The 'disc1-i386' file is highlighted.

Name	Size	Type	Modified	Attributes
dosutils		File Folder	4/8/01 11:36 PM	R
images		File Folder	4/8/01 11:38 PM	R
RedHat		File Folder	4/8/01 11:40 PM	R
.buildlog	1KB	BUILDLOG File	4/8/01 11:14 PM	R
.disc1-i386	1KB	DISC1-I386 File	4/8/01 11:40 PM	R
autorun	1KB	File	9/26/99 12:16 AM	R
boot.cat	2KB	Security Catalog	4/9/01 12:04 AM	R
Copying	18KB	File	9/7/99 1:02 PM	R
Readme	5KB	File	4/8/01 8:01 PM	R
RELEASE-NOTES	20KB	File	4/4/01 1:53 PM	R
RELEASE-NOTES.de	22KB	DE File	4/4/01 1:53 PM	R
RELEASE-NOTES.es	22KB	ES File	4/4/01 1:53 PM	R
RELEASE-NOTES.fr	23KB	FR File	4/4/01 1:52 PM	R
RELEASE-NOTES.it	21KB	IT File	4/4/01 1:53 PM	R
RELEASE-NOTES.ja	19KB	JA File	4/5/01 11:22 AM	R
RPM-GPG-KEY	2KB	File	9/25/99 2:20 PM	R
Trans.tbl	1KB	TBL File	4/9/01 12:04 AM	R

Figure 42. Contents of CD 1

Contents of 'Red Hat Linux_i3 [D:]'				
Name	Size	Type	Modified	Attributes
RedHat		File Folder	4/8/01 11:40 PM	R
Srpms		File Folder	4/9/01 12:03 AM	R
.disc2-i386	1KB	DISC2-I386 File	4/8/01 11:41 PM	R
autorun	1KB	File	9/26/99 12:16 AM	R
Copying	18KB	File	9/7/99 1:02 PM	R
Readme	5KB	File	4/8/01 8:01 PM	R
RPM-GPG-KEY	2KB	File	9/25/99 2:20 PM	R
Trans.tbl	1KB	TBL File	4/9/01 12:06 AM	R

Figure 43. Contents of CD 2

CD 3 - Linux Kickstart Server installation set, Drop 2.3:

- Download from:

<http://telorb.uab.ericsson.se/>

Select the link “Download Area” and then the “TSP4.0 Clusterware/OS” and the appropriate Drop.

The third CD will contain upgrades for RedHat 7.1 and the kickstart server software. Having untarred the download, burn the following tar files onto CD 3:

- tools.tar
- IO-KickStart-P1A18.tar, from IDP/D42
- RedHat-7.1-Upgrade-R1A01.tar, from IDP/D42

2. Install the software from the CDs on the kickstart server

http://telorb.uab.ericsson.se/presentation/projects/TelORB/Doc/R14_Doc/DocStartR14withLinux.htm

The software for the kickstart server is now installed. Before installing the TSP sotware from CD 3, the /opt directory on the kickstart was empty. It now contains four directories:

- TSP4.0/
- redhat/
- 3rdParty

- java

It's the contents of TSP that are of interest to us.

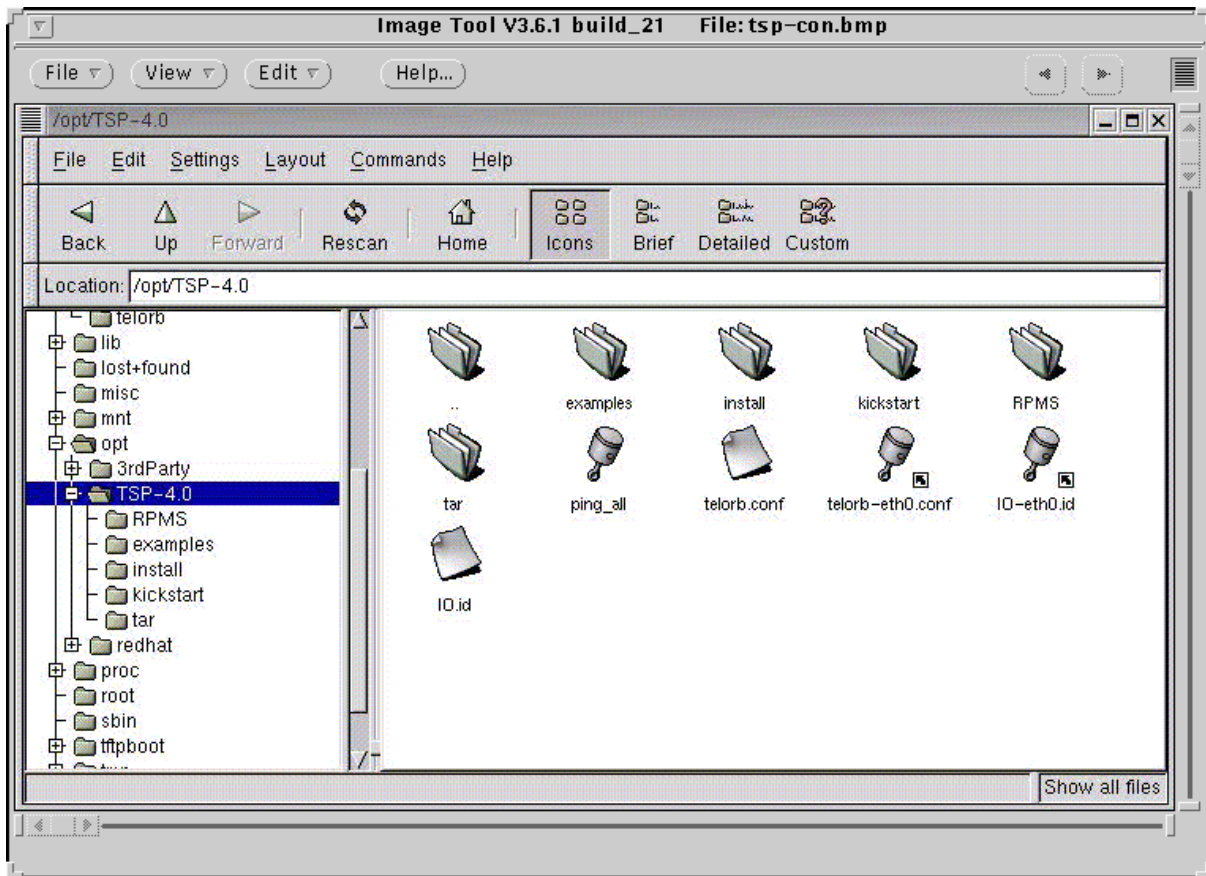


Figure 44. Contents of TSP 4.0

examples

Contains examples of telorb.conf files and scripts

install

Contains the ks-install script that guided you through the kickstart installation, and also a pxe tar file.

kickstart

Contains kickstart configuration files generated when we installed Linux and TSP Software:

e.g. ks-io-ide.cfg.

These files are read when the IO reboots.

RPMS

RedHatPacketManager, contains RHPM files (linux)

tar

Contains 'tools.tar' , copied in when installing CD 3.

IO.id

Contains number "1" by default.

telorb.conf

This file defines the local network configuration for the system. It also specifies which processors will run linux and which will run telorb.

3. Configure the 'telorb.conf' file on the kickstart server:

TSP Configuration File - telorb.conf :

'*telorb.conf*' is the single configuration file for the IO machines which configures all the daemons and services on the IO machines. This file is extended in TSP4.0 to contain information about the Linux traffic processors, the IO machines and further services.

To correctly edit this file it's necessary to obtain the following information from the system:

- MAC addresses of the IO processors.
- MAC addresses of the TelORB processors.
- Other network parameters which are necessary as input to the telorb.conf file.

The '*telorb.conf*' file must be edited to reflect the site-specific network-parameters and MAC-addresses. It must be saved with the name: '*telorb.conf*', on the KickStart server in the /opt/TSP4.0 directory.

This file, along with the Linux and TSP software is copied over to the IO machines during installation of the IO.

At a later stage in the configuration process, the '*telorb.conf*' file will be parsed to generate site-specific scripts used in the system.

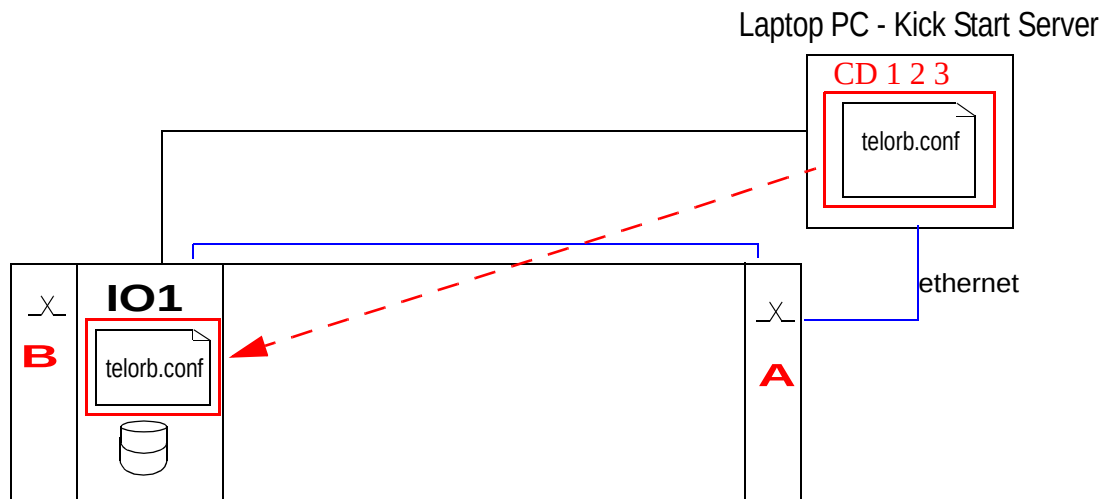


Figure 45. Copying of Software from Kickstart to IO

To find out network and IP information, on both IOs, type:

```
%> /sbin/ifconfig -a
```

Results of: /sbin/ifconfig -a:

```
[telorb@to26io1 init.d]$ /sbin/ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:A0:A5:4C:01:75
          inet addr:134.138.61.225  Bcast:134.138.61.255  Mask:255.255.255.128
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2821314 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3145135 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:12

eth1      Link encap:Ethernet  HWaddr 00:A0:A5:4C:01:76
          inet addr:134.138.62.225  Bcast:134.138.62.255  Mask:255.255.255.128
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:84822 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8494 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:11 Base address:0x2000

eth2      Link encap:Ethernet  HWaddr 00:80:37:8F:33:AE
          inet addr:134.138.187.105  Bcast:134.138.191.255  Mask:255.255.248.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1669046 errors:0 dropped:0 overruns:0 frame:0
          TX packets:22191 errors:0 dropped:0 overruns:0 carrier:0
          collisions:731 txqueuelen:100
          Interrupt:10 Base address:0x4000

eth3      Link encap:Ethernet  HWaddr 00:80:37:8F:33:AF
          inet addr:192.168.11.225  Bcast:192.168.11.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:3 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:12 Base address:0x6000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:5830 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5830 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
```

```
[telorb@to26io1 init.d]$
```

#####

```
[telorb@to26io2 /sbin]$ /sbin/ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:A0:A5:4C:01:5E
          inet addr:134.138.61.226  Bcast:134.138.61.255  Mask:255.255.255.128
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1495575 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1422973 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:12

eth1      Link encap:Ethernet  HWaddr 00:A0:A5:4C:01:5F
          inet addr:134.138.62.226  Bcast:134.138.62.255  Mask:255.255.255.128
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:457946 errors:0 dropped:0 overruns:0 frame:0
          TX packets:619369 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:11 Base address:0x2000

eth2      Link encap:Ethernet  HWaddr 00:80:37:8F:34:AE
          inet addr:134.138.187.106  Bcast:134.138.191.255  Mask:255.255.248.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1651510 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4077 errors:0 dropped:0 overruns:0 carrier:0
          collisions:125 txqueuelen:100
          Interrupt:10 Base address:0x4000

eth3      Link encap:Ethernet  HWaddr 00:80:37:8F:34:AF
          inet addr:192.168.11.226  Bcast:192.168.11.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:3 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:12 Base address:0x6000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:3075 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3075 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0

[telorb@to26io2 /sbin]$
```

Running the following command on an IO gives more details:

```
%> more /etc/hosts
```

```
[telorb@to26io1 examples]$ more /etc/hosts
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1localhost.localdomain localhost
#134.138.61.225 to26io1.uab.ericsson.se to26io1 io1 to26io1net0 io1net0
#134.138.62.225 to26io1.uab.ericsson.se to26io1 io1 to26io1net1 io1net1
#134.138.62.226 to26io2.uab.ericsson.se to26io2 io2 to26io2net1 io2net1
#134.138.61.226 to26io2.uab.ericsson.se to26io2 io2 to26io2net0 io2net0
134.138.61.225 to26io1net0.uab.ericsson.se to26io1.uab.ericsson.se to26io1 io1
to26io1net0 io1net0
134.138.62.225 to26io1net1.uab.ericsson.se to26io1.uab.ericsson.se to26io1 io1
to26io1net1 io1net1
134.138.62.226 to26io2net1.uab.ericsson.se to26io2.uab.ericsson.se to26io2 io2
to26io2net1 io2net1
134.138.61.226 to26io2net0.uab.ericsson.se to26io2.uab.ericsson.se to26io2 io2
to26io2net0 io2net0
```

The network can be represented as follows

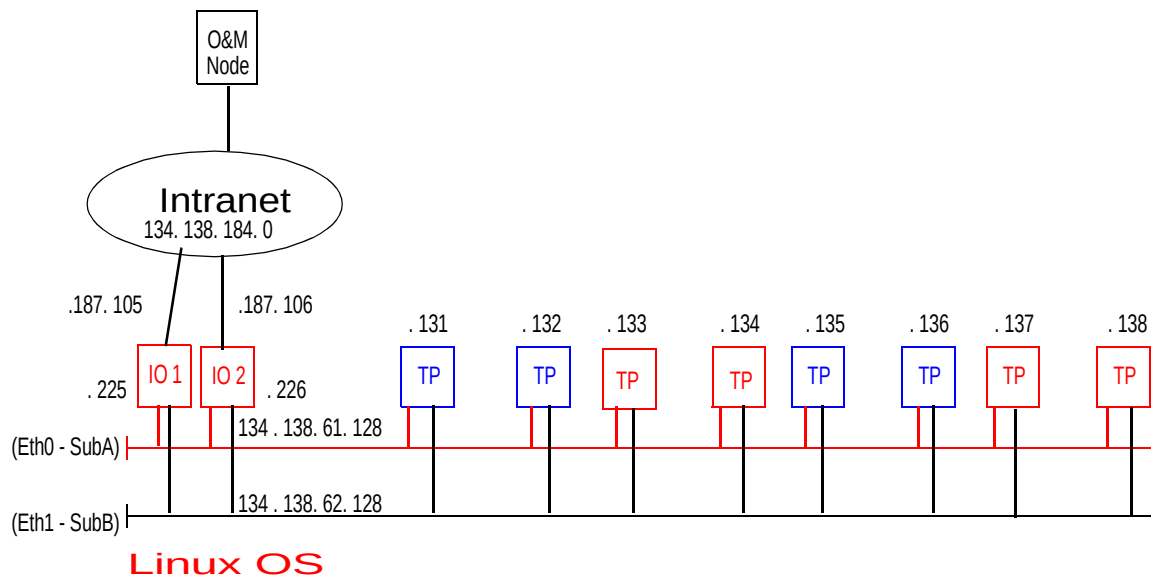


Figure 46. Our TSP system

Example of 'telorb.conf' using the results from above:

```
#####
# This configuration-file specifies all data used to run telorb-related
# services on IO-machines
#####
#eeimodr
#####
# define all timeservers; there may be more than one timeserver
# example:timeserver 134.138.77.26
# If both IO-machines are timeserver you should adress one through the
# first network, and the other through the second network.
#####
timeserver 134.138.61.225
timeserver 134.138.62.226

#####
# define the internal subnet, i.e. IO-processors and all dicosprocessors
# usually there are two internal networks
# To find out which interfaces you have and which networks are assigned
# to them type: "ifconfig -a" on you IO-machines
# Running the command on one IO-machine can give you the following output:
# to6io1:~>ifconfig -a
# lo0: flags=849<UP,LOOPBACK,RUNNING,MULTICAST> mtu 8232
#      inet 127.0.0.1 netmask ff000000
# eth0: flags=863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST> mtu 1500
#      inet 172.16.1.25 netmask fffffff0 broadcast 172.16.1.255
# eth1: flags=863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST> mtu 1500
#      inet 172.17.1.25 netmask fffffff0 broadcast 172.17.1.255
# eth2: flags=863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST> mtu 1500
#      inet 134.138.33.33 netmask fffff800 broadcast 134.138.39.255
# eth3: flags=863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST> mtu 1500
#      inet 134.138.187.29 netmask fffff800 broadcast 134.138.191.255
#
# Running the same on the other IO-machine gives you enough information.
# In the following example we assume that the interface-names are the
# same on IO1 and IO2.
#####
internal-network subnet0 134.138.61.128
internal-network netmask0 255.255.255.128
internal-network defaultgw0 134.138.61.129
internal-network bcast0 134.138.61.255
internal-network io1net0 134.138.61.225
internal-network io2net0 134.138.61.226
internal-network io1interfacename0 eth0
internal-network io1ethaddr0 00:A0:A5:4C:01:75
internal-network io2interfacename0 eth0
internal-network io2ethaddr0 00:A0:A5:4C:01:5E

internal-network subnet1 134.138.62.128
internal-network netmask1 255.255.255.128
internal-network defaultgw1 134.138.62.129
internal-network bcast1 134.138.62.255
internal-network io1net1 134.138.62.225
```

```

internal-network io2net1 134.138.62.226

internal-network io1interfacename1 eth1
internal-network io1ethaddr1 00:A0:A5:4C:01:76
internal-network io2interfacename1 eth1
internal-network io2ethaddr1 00:A0:A5:4C:01:5F

#####
# define how to boot the dicosprocessors
# use SMALLPROM or PXE
#####
booting PXE

#####
# define first port for terminal-server
# e.g. tsport 7001
#####
tsport 7001

#####
# define console-server ip-address
#####
consserverip 134.138.61.225

#####
# define all dicosprocessors, i.e. define the mac-address,
# ip-address
#
# if the processor shall be a loader the ip-address for the tftp-server
# has to be added (dicosprocessor1_0 eptftp_server_net0)
#####
dicosprocessor1_0 eth_addr_for_eth0 00:a0:a5:4c:01:4e
dicosprocessor1_0 ip_addr_for_eth0 134.138.61.131
dicosprocessor1_0 eptftp_server_net0

dicosprocessor1_1 eth_addr_for_eth1 00:a0:a5:4c:01:4f
dicosprocessor1_1 ip_addr_for_eth1 134.138.62.131
dicosprocessor1_1 eptftp_server_net1

dicosprocessor2_0 eth_addr_for_eth0 00:a0:a5:4c:01:58
dicosprocessor2_0 ip_addr_for_eth0 134.138.61.132
dicosprocessor2_0 eptftp_server_net0

dicosprocessor2_1 eth_addr_for_eth1 00:a0:a5:4c:01:59
dicosprocessor2_1 ip_addr_for_eth1 134.138.62.132
dicosprocessor2_1 eptftp_server_net1

dicosprocessor5_0 eth_addr_for_eth0 00:a0:a5:50:02:3f
dicosprocessor5_0 ip_addr_for_eth0 134.138.61.135

dicosprocessor5_1 eth_addr_for_eth1 00:a0:a5:50:02:40
dicosprocessor5_1 ip_addr_for_eth1 134.138.62.135

dicosprocessor6_0 eth_addr_for_eth0 00:a0:a5:50:02:23
dicosprocessor6_0 ip_addr_for_eth0 134.138.61.136

```

```

dicosprocessor6_1 eth_addr_for_eth1 00:a0:a5:50:02:24
dicosprocessor6_1 ip_addr_for_eth1 134.138.62.136

# dicosprocessor5_0 eth_addr_for_eth0 00:d0:b7:4e:18:f9
# dicosprocessor5_0 ip_addr_for_eth0 172.16.1.105

# dicosprocessor5_1 eth_addr_for_eth1 00:d0:b7:4e:19:04
# dicosprocessor5_1 ip_addr_for_eth1 172.17.1.105

#####
#
# nfs - mountpoints
#
#####
io1_share_nfs /opt/telorb/axe
io2_share_nfs /opt/telorb/axe

#####
#
# DicosApplogClient
# Collects applog messages from a TelORB system
# dicos_applog_client_debug_level - Debug printout level (0-silent, 9-verbose)
# dicos_applog_client_max_Mb_of_log - Limit file size to # mega byte
#
# dicos_applog_client_key
# The "applications" are a list of short application keys with the following
# restrictions:
#   - An application name may not be more than 30 characters long.
#   - Application names may only contain alphanumeric characters and underscore,
#     i.e; [A-Za-z0-9_].
# You can specify more than one key (see below).
#
#####
dicos_applog_client_debug_level 8
dicos_applog_client_max_Mb_of_log 4
dicos_applog_client_key HLR
dicos_applog_client_key SCP

#####
#
# VIP configuration
# specify the vip-network(s) and all vip_dicosprocessor(s)
#
#####
#vip-network vipnet0 192.168.63.0
#vip-network vipnetmask0 255.255.255.252
#vip_dicosprocessor4_2 eth_addr_for_vip0 00:d0:b7:4e:18:aa
#vip_dicosprocessor4_2 ip_addr_for_vip0 192.168.63.2

#vip-network vipnet1 192.168.64.0
#vip-network vipnetmask1 255.255.255.252
#vip_dicosprocessor4_3 eth_addr_for_vip1 00:d0:b7:4e:18:bb
#vip_dicosprocessor4_3 ip_addr_for_vip1 192.168.64.2

#vip-network vipnet2 192.168.63.4
#vip-network vipnetmask2 255.255.255.252

```

```
#vip_dicosprocessor5_2 eth_addr_for_vip2 00:d0:b7:4e:18:cc
#vip_dicosprocessor5_2 ip_addr_for_vip2 192.168.63.5
```

```
#vip-network vipnet3 192.168.64.4
#vip-network vipnetmask3 255.255.255.252
#vip_dicosprocessor5_3 eth_addr_for_vip3 00:d0:b7:4e:18:dd
#vip_dicosprocessor5_3 ip_addr_for_vip3 192.168.64.5
```

```
#####
#####
# Parameters used when running Network-redandancy
#####
#####
```

```
#
# define Transfer-Manager related stuff
#
#####
```

```
# List of hostnames or IP addresses to be use to
# access the TelORB (of the same zone) CosNaming IOR
telorb_cosnaming_host      134.138.61.131
telorb_cosnaming_host      134.138.61.132
```

```
# Identifier for the local zone
telorb_local_zone_id      1
```

```
# Identifier for the remote zone
telorb_remote_zone_id     2
```

```
#
# Inter Zone configuration. We have pairs consisting of one IP address
# and one target directory.
# Usually, the first IO machine of the remote zone has a target directory called
# loadingGroup01_1, the second loadingGroup01_2 and so on.
# The format and the programs are general and allow the usage of more than
# two IO machines in the future.
```

```
#
interzone_peer 134.138.187.105:loadingGroup01_1
interzone_peer 134.138.187.106:loadingGroup01_2
#interzone_peer 142.133.23.118:loadingGroup01_1
#interzone_peer 142.133.23.119:loadingGroup01_2
```

```
#####
#
# rsync configuration
#
#####
```

```
io1_rsync_path /opt/telorb/axe/loadingGroup01_1
io2_rsync_path /opt/telorb/axe/loadingGroup01_2
```

```
#####
#####
# Parameters used when running both Dicos and Linux-processors
```

```
#####
#####
#####
# define external subnets, i.e. the networks that are only connected
# to the IO-machine, but not to any Dicos or Linux traffic-processor
# use "ifconfig -a"
#####
#external-network subnet0 172.23.1.0

external-network netmask0 255.255.248.0
external-network defaultgw0 134.138.184.1
external-network bcast0 134.138.191.255
external-network io1net0 134.138.187.105
external-network io2net0 134.138.187.106
external-network io1interfacename0 eth2
external-network io2interfacename0 eth2

external-network subnet1 192.168.11.0
external-network netmask1 255.255.255.0
external-network defaultgw1 192.168.11.1
external-network bcast1 192.168.11.255
external-network io1net1 192.168.11.225
external-network io2net1 192.168.11.226
external-network io1interfacename1 eth3
external-network io2interfacename1 eth3

#####
# define all linuxprocessors, i.e. define the mac-address,
# ip-address
#####
linuxprocessor3_0 eth_addr_for_eth0 00:a0:a5:4c:00:54
linuxprocessor3_0 ip_addr_for_eth0 134.138.61.133
linuxprocessor3_0 eptftp_server_net0

linuxprocessor3_1 eth_addr_for_eth1 00:a0:a5:4c:00:53
linuxprocessor3_1 ip_addr_for_eth1 134.138.62.133
linuxprocessor3_1 eptftp_server_net1

linuxprocessor4_0 eth_addr_for_eth0 00:a0:a5:4c:00:78
linuxprocessor4_0 ip_addr_for_eth0 134.138.61.134
linuxprocessor4_0 eptftp_server_net0

linuxprocessor4_1 eth_addr_for_eth1 00:a0:a5:4c:00:79
linuxprocessor4_1 ip_addr_for_eth1 134.138.62.134
linuxprocessor4_1 eptftp_server_net1

linuxprocessor7_0 eth_addr_for_eth0 00:a0:a5:50:01:b2
linuxprocessor7_0 ip_addr_for_eth0 134.138.61.137
linuxprocessor7_0 eptftp_server_net0

linuxprocessor7_1 eth_addr_for_eth1 00:a0:a5:50:01:b4
linuxprocessor7_1 ip_addr_for_eth1 134.138.62.137
linuxprocessor7_1 eptftp_server_net1

linuxprocessor8_0 eth_addr_for_eth0 00:a0:a5:50:01:97
linuxprocessor8_0 ip_addr_for_eth0 134.138.61.138
```

```

linuxprocessor8_0 eptftp_server_net0

linuxprocessor8_1 eth_addr_for_eth1 00:a0:a5:50:01:98
linuxprocessor8_1 ip_addr_for_eth1 134.138.62.138
linuxprocessor8_1 eptftp_server_net1

#####
#
# VIP configuration
# specify the vip-network(s) and all vip_linuxprocessor(s)
#
#####
#vip-network vipnet4 192.168.63.0
#vip-network vipnetmask4 255.255.255.252
#vip_linuxprocessor6_2 eth_addr_for_vip0 00:d0:b7:4e:18:11
#vip_linuxprocessor6_2 ip_addr_for_vip0 192.168.63.3

#vip-network vipnet5 192.168.64.0
#vip-network vipnetmask5 255.255.255.252
#vip_linuxprocessor6_3 eth_addr_for_vip1 00:d0:b7:4e:18:22
#vip_linuxprocessor6_3 ip_addr_for_vip1 192.168.64.3

#vip-network vipnet6 192.168.63.4
#vip-network vipnetmask6 255.255.255.252
#vip_linuxprocessor7_2 eth_addr_for_vip2 00:d0:b7:4e:18:33
#vip_linuxprocessor7_2 ip_addr_for_vip2 192.168.63.7

#vip-network vipnet7 192.168.64.4
#vip-network vipnetmask7 255.255.255.252
#vip_linuxprocessor7_3 eth_addr_for_vip3 00:d0:b7:4e:18:44
#vip_linuxprocessor7_3 ip_addr_for_vip3 192.168.64.7

# (eof)

```

Configuring the IOs

Once the Kickstart Server installation is completed, and the 'telorb.conf' file is completed, the IO-Servers can be installed. The official documentation describing how to install the IO-processors using an external Linux KickStart server is found at:

http://telorb.uab.ericsson.se/presentation/projects/TelORB/Doc/R14_Doc/DocStartR14withLinux.htm

You need the following equipment

- A kickstart server installed according to previous instructions
- A Cat5 Ethernet cable
- A 9-pin serial null-modem cable.

Minicom on the Kickstart Server

Official documentation is found at:

http://telorb.uab.ericsson.se/presentation/projects/TelORB/Doc/R14_Doc/DocStartR14withLinux.htm

Minicom is a communication program which allows the Linux kickstart server communicate with the serial communications port of an IO or Traffic processor. The minicom utility establishes a full-duplex terminal connection to a remote host. Once the connection is established, a remote session using minicom behaves like an interactive session on the local terminal.

Installing Linux on IO 1

Official documnetation is found at:

http://telorb.uab.ericsson.se/presentation/projects/TelORB/Doc/R14_Doc/DocStartR14withLinux.htm

Start a minicom session on the kickstart server as. Connect a "Null Modem" cable from the serial port on the KickStart processor with Linux OS, to the serial port on the 'target' IO processor.

When the installation is successfully completed, let the target IO processor perform a complete reboot. This IO-processor is now installed with the following default users and passwords:

- 'root' user with a password 'rootroot'
- 'telorb' user with a password 'tel123orb'

The kickstart server has loaded the following software on to the IO:

- A copy of RedHat 7.1 CD 1 & 2, plus patches.
- A DHCP server
- A PXE-server
- A TFTP-server
- A NTP-server
- A NFS-server
- TSP4 KickStart Installation Files
- Miscellaneous configuration files.

Installing Linux on IO 2

The installation is done in exactly the same way as IO1 with the only difference being IO1 will act as a KickStart server for IO2.

For simple hardware testing, the 'ping_all' script can be run on both IOs to test network connectivity to all processors within the node:

```
%> cd /opt/TSP4.0
%> ./ping_all
```

If the IO-processors didn't boot correctly there may be a problem with the PXE server or the DHCP server running on the KickStart. It may be necessary to restart the PXE server and/or the DHCP server on the KickStart.

The roles of the IO processors

The roles of the IO processors are primarily:

- to execute O&M functions
- to provide a file-system. TelORB / Linux processors don't have a file-system of their own. You cannot store a file on a TelORB processor. Instead the IO processors use NFS to act as file servers. All required files are stored on the IO processors and accessed later using NFS.
- NTP (Network Time Protocol) Server
- DHCP-server (Dynamic Host Configuration Protocol).
- PXE-server
- TFTP-server (Trivial File Transfer Protocol).
- Software Terminal Server
- System Log
- Error Dump Receiver
- Transfer Manager
- Application Logger
- Crash Collector

To play these roles successfully requires special configuration.

DHCP-server

This service allocates IP-addressing information to the processors when they are booted. This is handled by dhcp-daemon processes executing on both IO-processors. This process is started automatically on the IO-processors. The DHCP configuration is found in the /opt/telorb/etc/dhcpd.pxe.conf file. The main information is the list of IP-addresses to be handed out when receiving DHCP-requests.

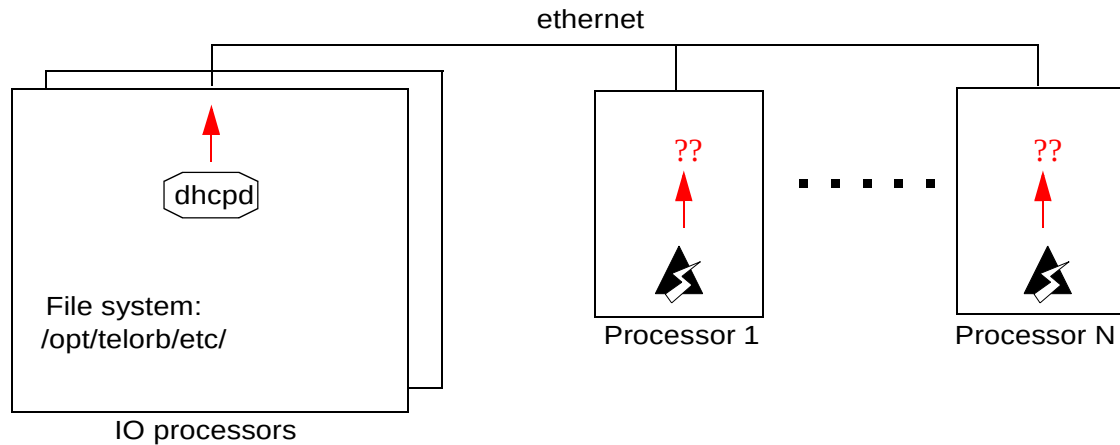


Figure 47. DHCP request and Server

Pxe-Server:

When the traffic processors boot, they boot as pxe clients. The pxe server on the IO responds, using the 'pxe.cf' file to load the appropriate OS on each processor.

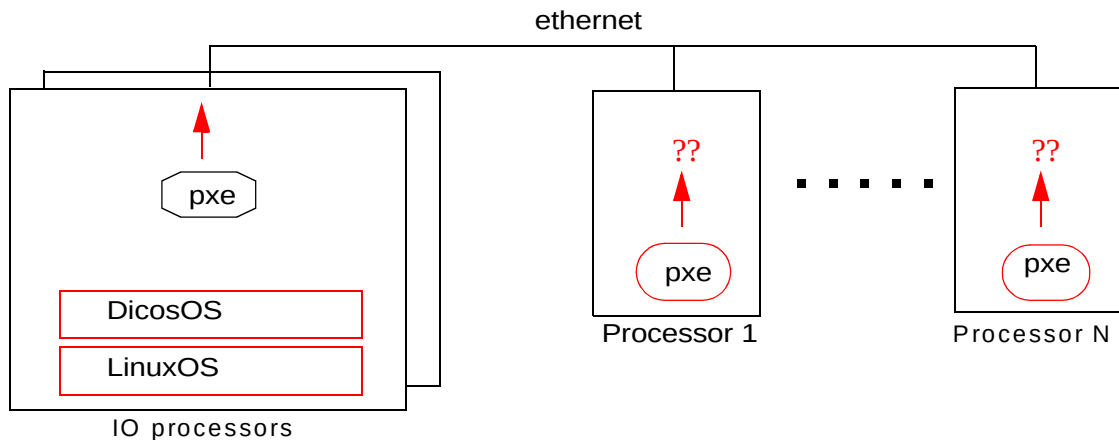


Figure 48. PXE requests and server

TFTP-server

This service is used to transfer files between the IO-processors and the other processors. It is used e.g. for file transfer during early start phases and transfer of error dumps. The TFTP service is provided by a tftp-daemon process executing on the IO-processors. TFTP configuration data is found in the /opt/telorb/etc/ep_tftpd.conf file. It contains IP-addresses of the TelORB processors configured to be loaders and all the Linux processors.

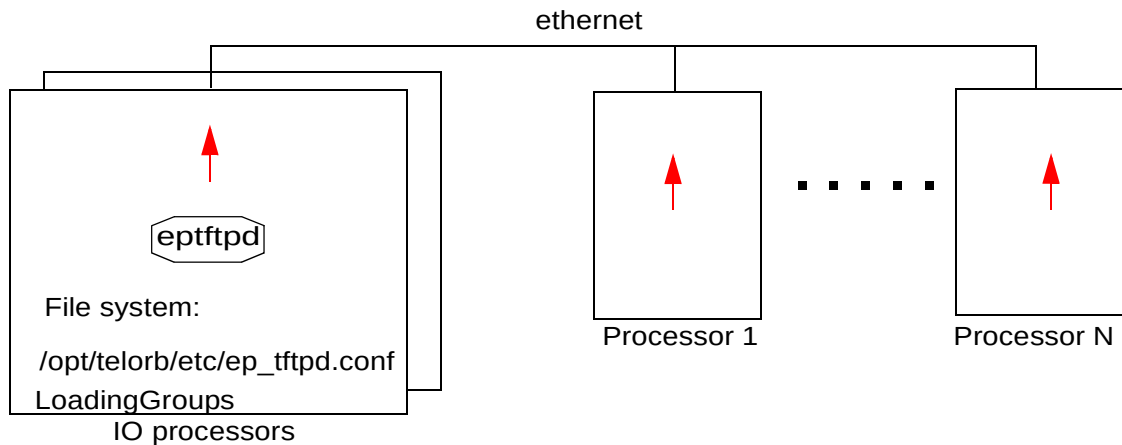


Figure 49. TFTP server

Providing a File System

The filesystem is provided to the TelORB processors via NFS. An entire new distributed filesystem is included in R14. The filesystem is used by the operating system itself to perform file access operations. NFS mounting is used in the system to access the file systems on the IOs. An initial mounting of directories must be manually specified.

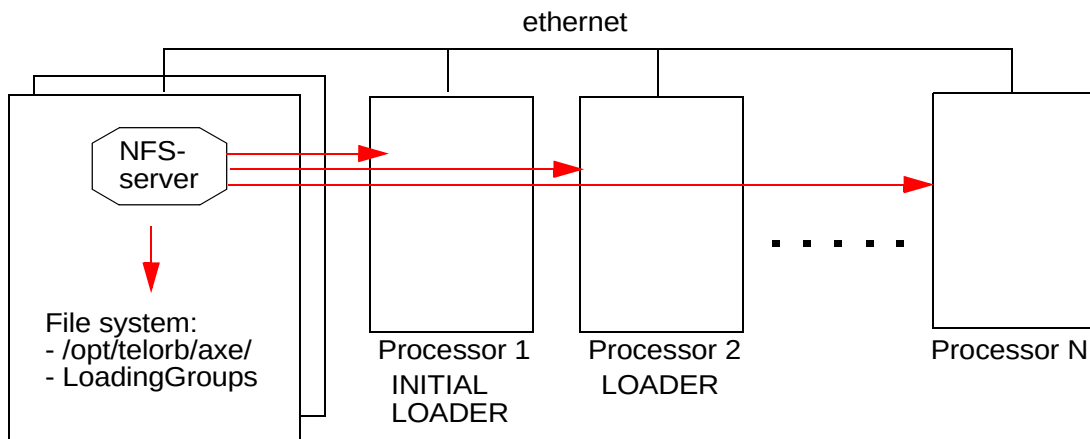


Figure 50. FileSystem via NFS

Software Terminal Server

Console printouts from any traffic processor can be received by telnetting to the Software Terminal Server (STS). It executes on IO processor 1 only. You telnet to port 7001 for processor 1, 7002 for processor 2 and so on. This service requires parameters to be set in the telorb.con configuration file, using the parameter 'tsport'. Only one client is allowed to connect to each port.

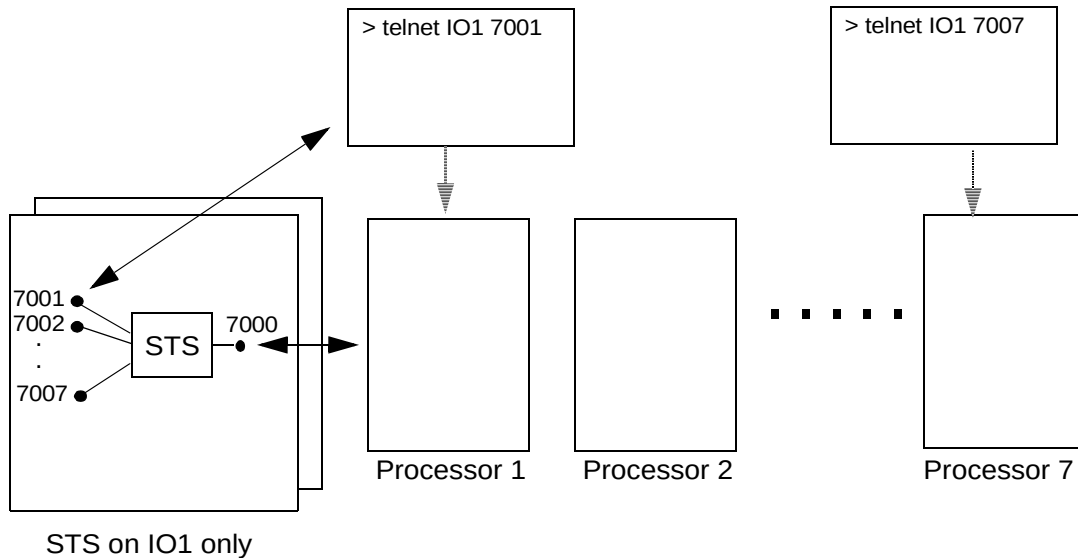


Figure 51. STS consoles

System Log

If the parameter '-log' is entered when starting STSsw, the STS places processor log files on one of the IO processors below the directory /opt/telorb/consolelogs/.

Application Log

The new application log feature allows an application to send log-messages to a client process on the IO-machine that writes the message to a log file. Different applications can tag their log-messages to end up in different log files on the IO-machines. Some of the advantages are:

- Just one log-file for the entire application, not one per traffic processor.
- Log-messages are time-stamped.

Transfer Manager

The Transfer Manager is a part of the improvements that have been made to the network redundancy functionality in R14. The functionality previously included in the Jambala "scheduler" has now been re-implemented and integrated into TelORB. The Transfer Manager is responsible for transferring backups from the IO processors in one zone to the IOs in another, in order to execute "external synchronisations" and reconciliations.

Error Dump Receiver

TelORB can generate several types of dumps when error situations occur. The 'dumpd' process runs on the IO processors and collects the dumps, placing them in the directory /opt/telorb/dumps.

Crash Collector

The Crash Collector is a function/daemon which runs independently on both IO processors. This collects maintenance and troubleshooting information should a severe "event" happen in TelORB.

Every 5 minutes, it scans the /opt/telorb/dumps directory and checks for new dumps that may have occurred. Should it detect a dump of a particular severity it will collect up a number of files which are relevant for troubleshooting. The generated archives are placed in the directory /export/home/telorb/crashcollector.

Site-specific Scripts

When the IO is being configured, the 'telorb.conf' file, along with the Linux and TSP software is copied over to the IO machine.

When the IO reboots a file generated by the kickstart installation:

'ks-io-ide.cfg', is used. This calls two important scripts:

'prepareTelORB.sh' and 'configureTelORB.sh'

```
%> cd /opt/telorb/codearchive/iom_scripts
```

prepareTelORB.sh

This script is run during the IO installation and is used to set up permissions regarding access to directories on the IO. If the telorb.conf file is edited, this script needs to be run again. This can be done manually or will be run automatically by rebooting the IO. If running the script manually, you must be logged in as 'su'.

configureTelORB.sh

This script is also run during the IO installation and creates new start_services, stop_services and other scripts. If the telorb.conf file is edited, this script needs to be run again. This can be done manually or will be run automatically by rebooting the IO. If running the script manually, you must be logged in as 'telorb'.

The scripts generated contain configuration specific values for TFTP, DHCP services and all the remaining services like pxe, dumps etc.

```
%> cd /opt/telorb/etc
%> ls
```

```
dhcpd.pxe.conf
start_services
stop_dhcp
pxe.cf
status_dhcp
stop_services
ep_tftpd.conf
start_dhcp
status_services
telorb.conf
ntp.conf
rsyncd.conf
transfermgr.conf
dhcpd.leases
ntpstep-tickers
crashcollector.conf
```

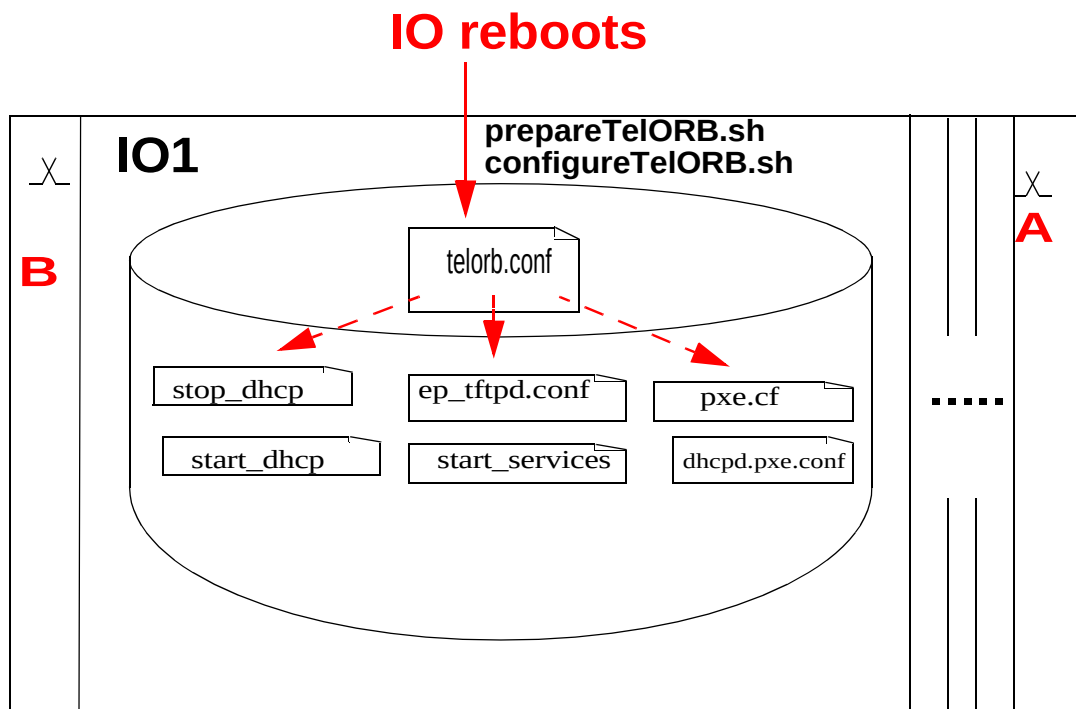


Figure 52. Script generation upon reboot of IO

Startup of the IO processor

The start_services script is found in the */opt/telorb/etc* directory. Below is the start_services script that has been generated from our configuration file.

```
%> [telorb@to26io1 etc]$ pwd
%> /opt/telorb/etc
%> [telorb@to26io1 etc]$ more start_services

#!/bin/csh -f
source /opt/telorb/codearchive/Unix_IDP/env/setup.csh

cd /opt/telorb/codearchive/iom_scripts
if(`./checkGroup.sh telorb` != "") then
    echo "To run this script you need to be member of the group telorb."
    exit
endif
setenv DUMPDIR /opt/telorb/dumps

# Settings for ep-tftp
setenv DICOS_FILESYSTEM_ROOT /opt/telorb/axe

## Start the telorb services

./startDaemon.sh ./guardDs -th 134.138.61.131 -th 134.138.61.132 -th
134.138.61.133 -th 134.138.61.134 -th 134.138.61.137 -th 134.138.61.138 -hbt 10
-hbi 5 -rci 5 > /dev/null
./startDaemon.sh ./guardEptftp /opt/telorb/etc/ep_tftpd.conf > /dev/null
./startDaemon.sh ./guardFsi -n io1 > /dev/null
./startDaemon.sh ./guardCrashCo -i 134.138.61.225 -a 134.138.61.225 -b
134.138.61.226 -p -j -c /opt/telorb/etc/crashcollector.conf
./startDaemon.sh ./guardDump -s $DUMPDIR -i
./startDaemon.sh ./guardSts 8 7001 -log
./startDaemon.sh ./guardSysC -L -d 5 -m io1 -t 134.138.61.131-136
./startDaemon.sh ./guardAppC -s 4M -d 8 -m io1 -t 134.138.61.131-136 HLR SCP >
/dev/null
./startDaemon.sh ./guardTrMgr /opt/telorb/etc/transfermgr.conf
./startDaemon.sh ./guardPxe -d -f /opt/telorb/etc/pxe.cf
./startDaemon.sh ./guardMirror -s 20 -i 134.138.61.226 -l 1
[telorb@to26io1 etc]$
```

To start the services enter the following:

```
%> ./start_services
%> ./start_dhcp
```

start_services: starts the services without closing down any that are already running

stop_services: stops all of the running services.

This is what the stop_services script looks like:

```
%> [telorb@to26io1 etc]$ pwd
%> /opt/telorb/etc
%> [telorb@to26io1 etc]$ more stop_services

#!/bin/sh
## Stop all telorb services

#set -x

cd /opt/telorb/codearchive/iom_scripts

./killProc.sh "\./guardFsi"
./killProc.sh "\./guardDs"
./killProc.sh "\./guardEptftp"
./killProc.sh "\./guardDump"
transferjava=`./childProcId.sh \./guardTrMgr`
./killProc.sh "\./guardTrMgr"
./killProc.sh "\./guardCrashCo"
./killProc.sh "\./guardPxe"
./killProc.sh "\./guardSysC"
./killProc.sh "\./guardAppC"
./killProc.sh "\./guardMirror"
./killProc.sh "\./guardSts"
filename=`./getIoBinPath STSsw D14`
./killProc.sh `/bin/basename $filename`
filename=`./getIoBinPath FileServerInterpreter AXG428`
./killProc.sh `/bin/basename $filename`
filename=`./getIoBinPath DUCKServer MD12`
./killProc.sh `/bin/basename $filename`
filename=`./getIoBinPath TftpDumpd MD44`
./killProc.sh `/bin/basename $filename`
filename=`./getIoBinPath EpTftpd MD41`
./killProc.sh `/bin/basename $filename`
filename=`./getIoBinPath DicosSyslogClient D35`
./killProc.sh `/bin/basename $filename`
filename=`./getIoBinPath DicosApplogClient D81`
./killProc.sh `/bin/basename $filename`
filename=`./getIoBinPath TransferManager D109 jar jar`
./killProcById.sh `/bin/basename $filename` $transferjava
filename=`./getIoBinPath CrashCollector D121 scripts sh`
./killProc.sh `/bin/basename $filename`
filename=`./getIoBinPath Pxe D48`
./killProc.sh `/bin/basename $filename`
filename=`./getIoBinPath IoMirror D156 scripts sh`
./killProc.sh `/bin/basename $filename`
sleep 2

## tidy up

[telorb@to26io1 etc]$
```

To find out the status of the services in the machine, run the following script:

```
%> [telorb@to26io1 etc]$ pwd
%> /opt/telorb/etc
%> [telorb@to26io1 etc]$ ./status_services

guardFsi (pid 19977) is running ...
guardDs (pid 19899) is running ...
guardEptftp (pid 19935) is running ...
guardDump (pid 20101) is running ...
guardSysC (pid 20217) is running ...
guardAppC (pid 20286) is running ...
guardTrMgr (pid 20358) is running ...
guardCrashCo (pid 20043) is running ...
guardPxe (pid 20432) is running ...
guardMirror (pid 20520) is running ...
guardSts (pid 20141) is running ...
STSw.LM_D14.R1I02.i486-Redhat-6.0 (pid 20338) is running ...
FileServerInterpreter.LM_AXG428.R1I03.i486-Redhat-6.0 (pid 20097) is running ...
DUCKServer.LM_MD12.R1I03.i486-Redhat-6.0 (pid 20022) is running ...
TftpDumpd.LM_MD44.R1J.i486-Redhat-6.0 (pid 20261) is running ...
EpTftpd.LM_MD41.R1I03.i486-Redhat-6.0 (pid 4836 20056 20055) is running ...
DicosSyslogClient.LM_D35.R1J01.i486-Redhat-6.0 (pid 20469) is running ...
DicosApplogClient.LM_D81.R1B01.i486-Redhat-6.0 (pid 20403) is running ...
TransferManager.LM_D109.R1A15.jar (pid 20456) is running ...
CrashCollector.LM_D121.R1B01.sh (pid 20114) is running ...
Pxe.LM_D48.R1C.i486-Redhat-6.0 (pid 20573) is running ...
IoMirror.LM_D156.R1B01.sh (pid 20568) is running ...

%> [telorb@to26io1 etc]$ ./status_dhcp
guardDhcp (pid 5833) is running ...
Dhcp.LM_D54.R1J05.i486-Redhat-6.0 (pid 5865) is running ...
[telorb@to26io1 etc]$
```

Loading the Cluster

Rebooting the IOs generates new scripts in /opt/telorb/etc that are needed to load the cluster. Once the IOs have rebooted they're ready to load the appropriate software in the cluster. To achieve this all processors need to be rebooted simultaneously, i.e. an Initial Start.

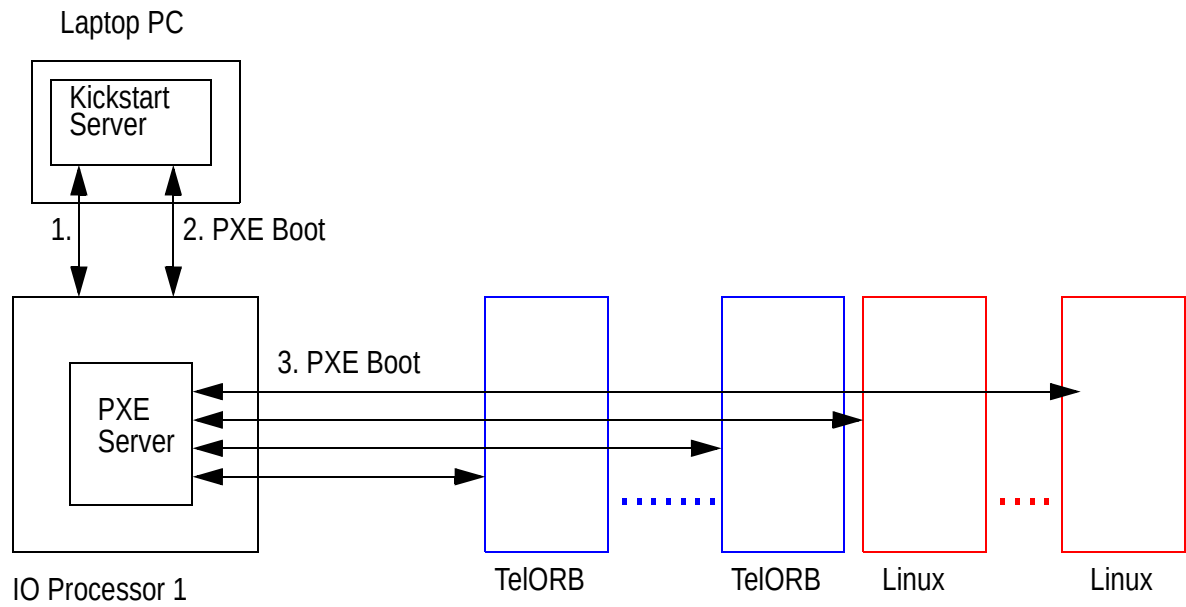


Figure 53. Virgin installation of a cluster

The start procedure is a series of boot procedures divided into different phases. In each phase more and more functionality is added to the processors by fetching the required code from the IO-processors. Then the processors will start executing the fetched code. The added functionality makes it possible to reach the next phase in the start/restart procedure. When the last phase is completed, the system is up and running.

All processors in the cluster are booted with pxe functionality. PXE, or Pre-Boot Execution Environment, is a specification published by Intel that allows a machine to boot itself from its network adapter without user intervention.

Start phases

The start phases are described separately for dicos and linux processors.

Start phases - Dicos

The following four start phases exist (see figure below):

- OSloader
- Initial
- Base Functionality
- Total Functionality

First loadmodules belonging to each start phase must be loaded. (Some loadmodules contains code others contains configuration data.) Then

the functionality of the start phase is started. For earlier start phases (OSloader, Initial) often includes drivers being started. Starting later phases (base or total functionality) implies starting Dicos processes by calling the start and systemStarted methods.

Loading is carried out via different protocols for different start-phases. If the processor is initial-loader, loader or non-loader also effects which loading protocol is used for a certain start phase.

OSloader

The BIOS of all processors is configured to boot from the network in a so called PXE (Pre Execution Environment) boot. The PXE boot includes obtaining IP-address information via DHCP and loading the OSloader program from the IO-processors. Each processor is loaded individually in this phase.

INITIAL

This start-phase is loaded individually via the TFTP protocol for initial loader and loader. For non-loaders this phase is loaded via a TelORB specific multicast loading controlled by the initial loader or loader.

BASE

This start-phase is loaded individually via the TFTP protocol for initial loader and loader. All non-loaders are loaded via a TelORB specific multicast loading controlled by the initial loader or loader.

TOTAL

In this phase the loading is controlled by the initial loader or loader communicating with the IO via NFS. All non-loaders are loaded via a multicast load controlled by the initial loader. The loader takes part in the multicast also as a receiver .

TOTAL (NFS)	TOTAL (NFS)	TOTAL (Multicast)	TOTAL (NFS)
BASE (TFTP)	BASE (TFTP)	BASE (TFTP)	BASE (TFTP NFS)
INITIAL (TFTP)	INITIAL (TFTP)	INITIAL (TFTP)	Linux+TelORB Init. (PXE)
OSloader (PXE)	OSloader (PXE)	OSloader (PXE)	RAM-disc (PXE)
INITIAL LOADER	LOADER	NON LOADER	LINUX

Start phases - Linux

The following four start phases exist (see figure above):

- RAM-disc
- Linux+TelORB Initial
- Base Functionality
- Total Functionality

Processors configured to run on Linux are loaders of themselves. Thus all start phases are loaded individually.

RAM-disc

The BIOS of all processors is configured to boot from the network in a so called PXE (Pre Execution Environment) boot. The PXE boot includes obtaining IP-address information via DHCP. A RAM-disc is installed on the Linux processors.

Linux+TelORB Initial

The Linux kernel, the rest of linux and some “initial” TelORB functionality is loaded via PXE. All of it is installed on the RAM-disc.

Base Functionality

By using the “initial” TelORB functionality, each processor loads its bootload table via tftp. The bootload table contains the information about what loadmodules should be loaded in the remaining part of the start. These loadmodules are then loaded via NFS and installed on the RAM-disc.

TOTAL Functionality

All loadmodules belonging to the total functionality (i.e application loadmodules) are loaded via NFS (Network File System) and the corresponding functionality is started.

Initial Start

The initial start includes both dicos and linux processors. The base functionality started and total functionality started is synchronized in the cluster, regardless what operating system the processors execute. However, the details about what is done in each start phase is different for dicos and linux processors. Therefore the initial start is described separately for dicos and linux.

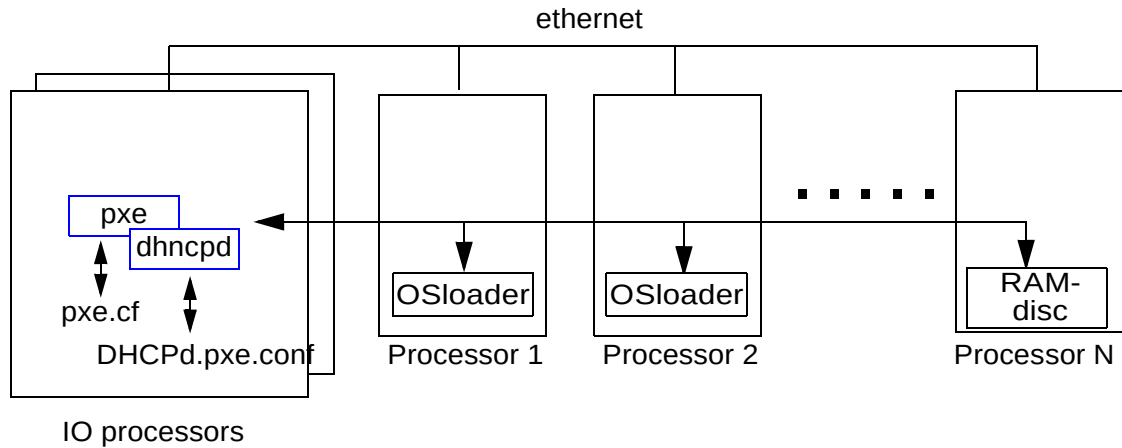
Initial Start (Dicos)

OSloader

Every processor is configured to boot from the network in a PXE boot. All processors will broadcast a DHCP request (see figure below). The request will be answered by the dhcp-daemon process executing on the IO processor. Via DHCP signalling each processor will be informed about its own IP-address, netmask etc. This information is fetched from

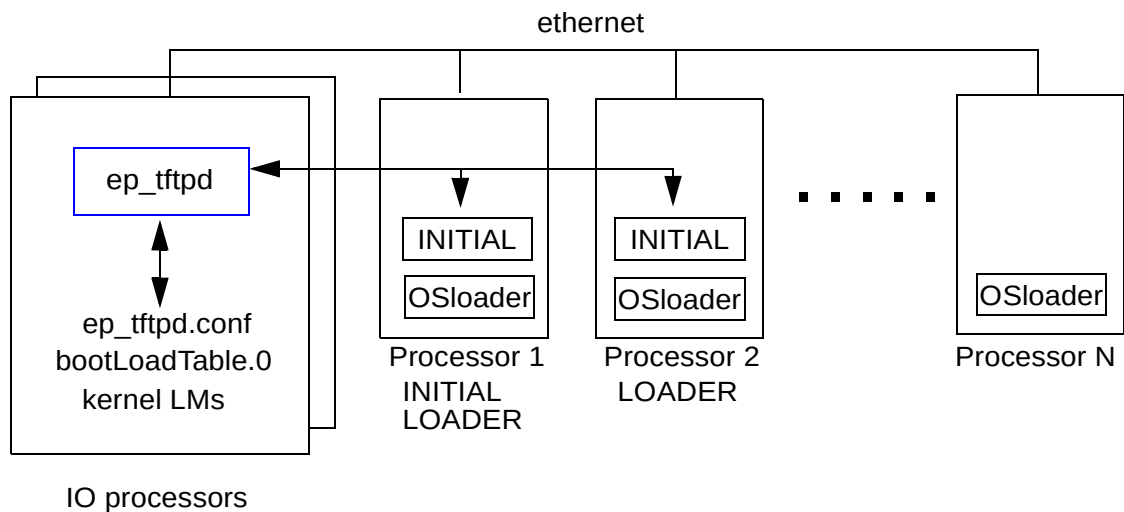
a configuration file located in the /opt/telorb/etc directory of the IO processor that happens to answer the broadcasted request.

After that the processors will receive their boot images based on the information in pxe.cf (different for dicos and linux). The dicos processors will get an OSloader program. Dicos processors will load and start an OSloader.



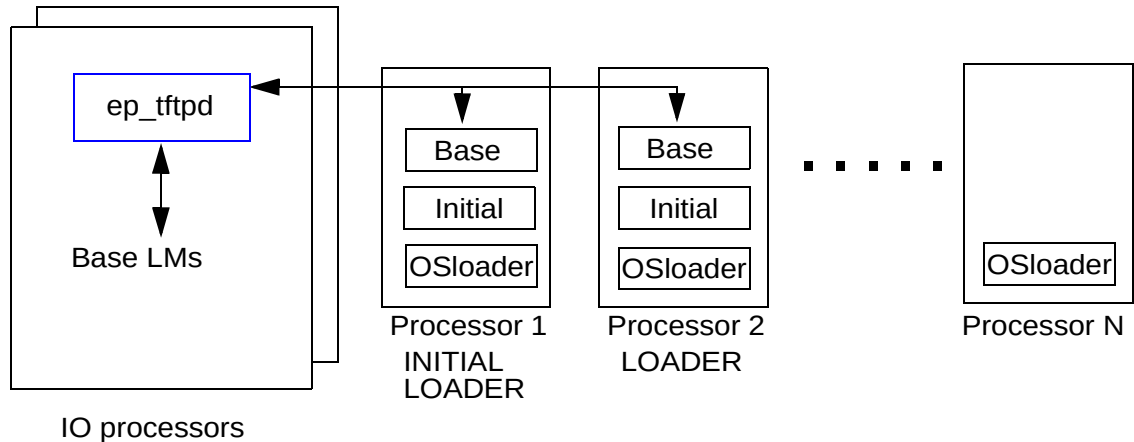
INITIAL (loaders):

All processors running dicos now tries to connect to a boot loading device. At this stage the only dicos processors that will succeed to connect to a loading device are the loaders. They will be accepted by the ep_tftpd process executing on the IO processor because they are mentioned in the /opt/telorb/etc/ep_tftpd.conf file. A TftpFileBoot of the loaders will take place, while the non-loaders will stop and wait until a loader is ready to start the loading of the non-loaders. The loaders will first load their bootLoadTables which are the basis for all further loading of these processors. After that the Dicos kernel is loaded.



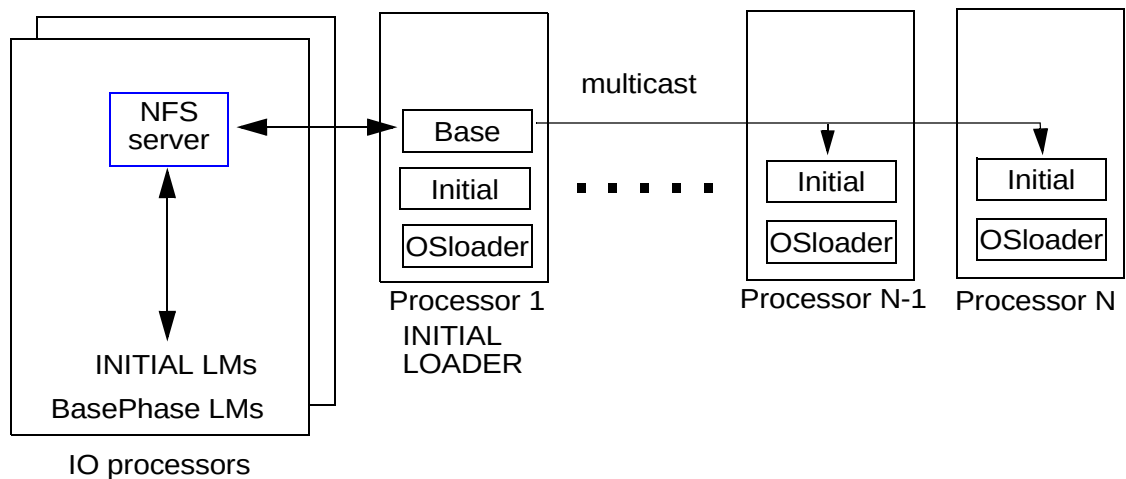
Base Functionality partly (loaders):

The loaders will now go on loading the base phase loadmodules. The database is started locally on the loaders. The initial loader will take on the role as Configuration Manager. It will read the information in `axe/loadingGroup01/initialData` i.e `earlyBaseConfigurationDataLM` and `baseConfigurationDataLM` and store corresponding information in the database. At this stage the database is only local (not distributed).



Loading Initial and Base (non loaders):

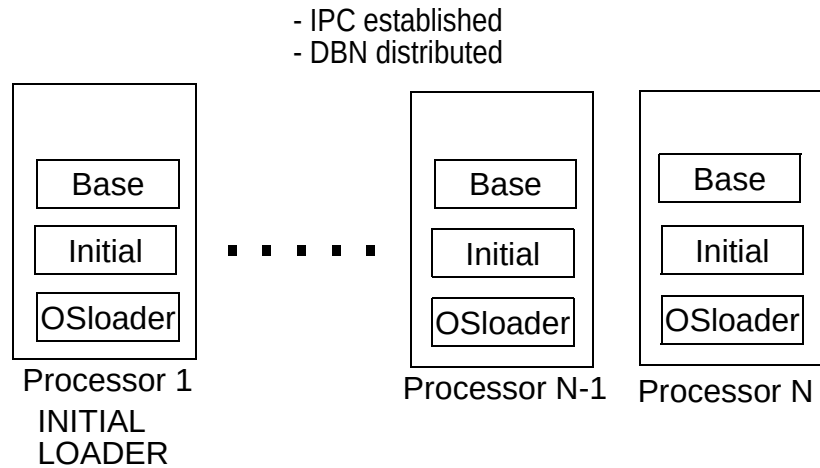
An LM distributor is started on the loader. It establishes a loadlink via ethernet to all non-loader processors. This is referred to as the multicast loadlink. The LM distributor reads the LMs from the filesystem on the IO processor via NFS. It distributes the Kernel LMs (including bootloadtables for the non-loaders) and the Base Phase LMs. The kernel is started on the non-loaders.



Base Functionality complete

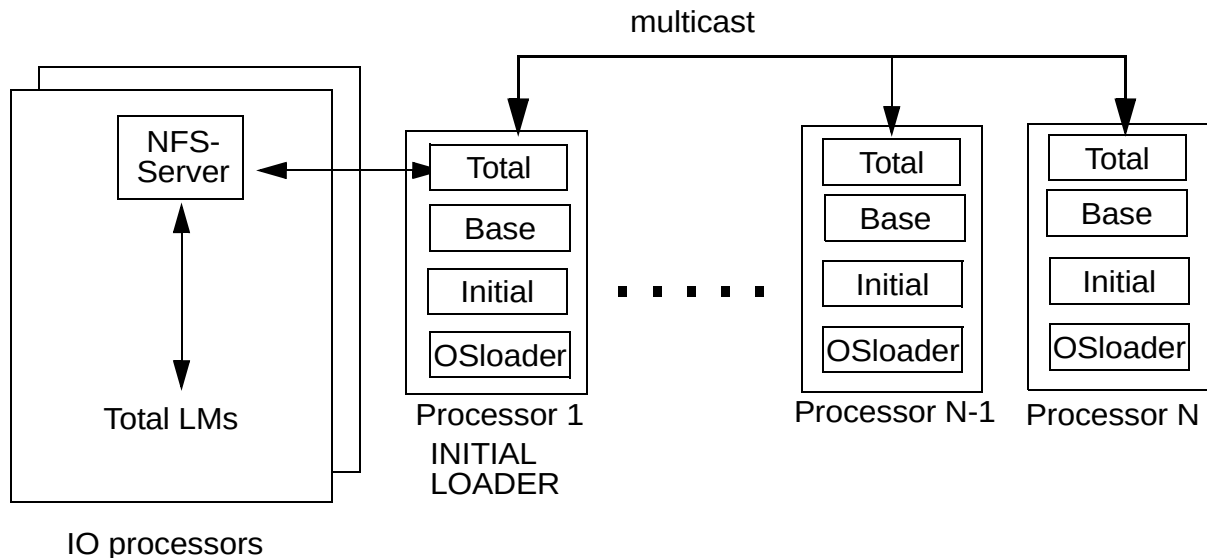
Now IPC (Inter Process Communication) is established. All processors connects to the Configuration Manager (CM) executing on the initial

loader. The second loader is stand-by Configuration manager. The non-loaders have CM-agents executing on them. The configuration manager takes care of any reconfigurations, planned or unplanned, and updates the system so it can continue in the new configuration. The database goes distributed. This completes the start of the base functionality.



Total Functionality:

First the total phase loadmodules are loaded. These are the loadmodules of the application. They are loaded via a multicast loadlink for all dicos processors including the loaders. The secondary loader will setup a separate multicast loadlink. Only one processor participates in this second multicast load, the secondary loader itself. Finally the total functionality is started which implies that the system is up and running.



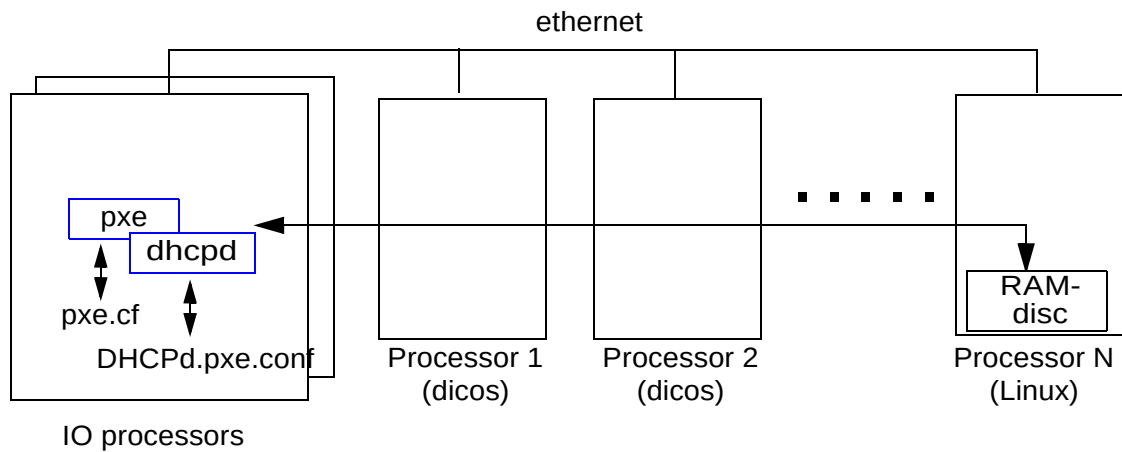
Initial Start (Linux)

Please note that the initial start of the linux processors is executed in parallel with the initial start of the dicos processors. Base Functionality started and Total Functionality started are synchronized between dicos and linux processors.

RAM-disc

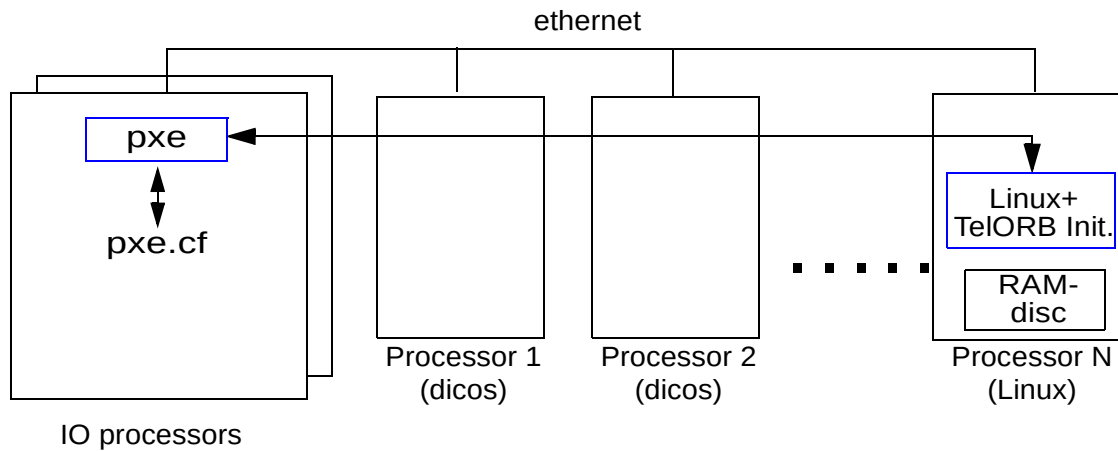
Also the linux processors are configured to boot from the network in a PXE boot. They will broadcast a DHCP request (see figure below). The request will be answered by the dhcp-daemon process executing on the IO processor. Via DHCP signalling each processor will be informed about its own IP-address, netmask etc. This information is fetched from a configuration file located in the /opt/telorb/etc directory of the IO processor that happens to answer the broadcasted request.

After that the processors will receive their boot images based on the information in pxe.cf (different for dicos and linux). The linux processors will receive a boot image that installs a RAM-disc where linux eventually will be installed.



Linux+TelORB Initial:

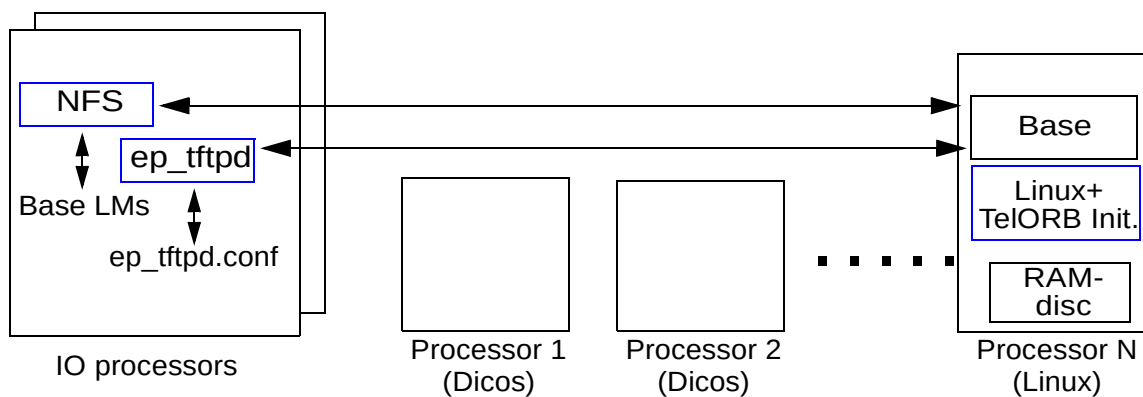
The PXE boot continues by loading the linux kernel, the rest of linux and some “initial” TelORB functionality. All of it is installed on the RAM-disc.



Base Functionality:

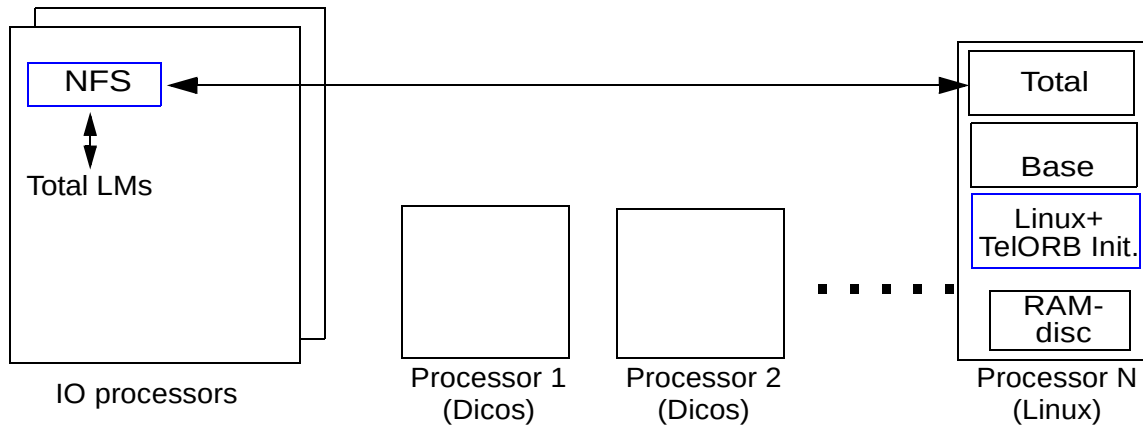
By using the previously loaded “initial” TelORB functionality, the processors are able to load their bootLoadTables which is the basis for all further loading of these processors. The linux processors will be accepted by the ep_tftpd process executing on the IO processor, because they are all mentioned in the /opt/telorb/etc/ep_tftpd.conf file. The loadmodules mentioned in the bootload tables are then loaded via NFS and installed on the RAM-disc.

When the base functionality is started, IPC (Inter Process Communication) is established. The CM-agent on all processors connects to the Configuration Manager (CM) executing on processor 1. The configuration manager takes care of any reconfigurations. This completes the start of the base functionality.



Total Functionality:

First the total phase loadmodules are loaded. These are the loadmodules of the application. They are loaded individually via NFS for all linux processors. Finally the total functionality is started.



OSLoader and pxe.cf

There are 2 Operating Systems in the system

- Dicos- Dicos runs on both loader processors and some or all of the other (Traffic) processors
- Linux - There are two configurations of Linux in the system:
 - The IOs run a server configuration
 - Linux may also run on some of the Traffic Processors

When the processors pxe boot, the IO uses the pxe.cf file to determine the appropriate OS to load on each processor. The dhcpd.pxe.conf file includes the MAC addresses for all processors, the MAC addresses then determine the IP addresses for each processor. Please note that the files dhcpd.pxe.conf and pxe.cf are generated from the telorb.conf file.

According to the pxe.cf file the pxe server supports three types of boot:

```
.
.
# Extract from pxe.cf
# Boot server types supported
[Service_Types]
14,Dicos
13,linux-tp
12,linux-install
.
.
```

The IP address from the dhcpd.pxe.conf file is used to determine which OS to load.

▪

▪

#Extract from pxe.conf

[X86PC/UNDI/MENU]

14,Dicos pxe boot

13,Linux pxe boot [133, 134, 137, 138]

12,Linux install boot [225, 226]

▪

▪

- Processors with IP addresses ending in 225 and 226 will be reloaded as an IO configuration.
- Processors with IP addresses 133, 134, 137 and 138 will be loaded with the Linux OS.
- The default boot is a 'Dicos pxe boot'. All other processors will be loaded with the TelORB OS.

dhcpcd.pxe.conf

```
# this is the dhcpcd configuration file used by
# the ISC version 3 dhcpcd.

# define site specific options
option epTftpdServerAddress    code 152 = ip-address;
option dumpServerAddress      code 153 = array of ip-address;
option consoleServerAddress    code 154 = array of ip-address;
option consoleServerPort      code 155 = unsigned integer 16;
option telorbIpcDevice         code 156 = unsigned integer 8;
option telorbHostsData         code 157 = text;

option time-offset 3600;

# Ip address of timeservers
option time-servers 134.138.62.226, 134.138.61.225;
option ntp-servers 134.138.62.226, 134.138.61.225;

option dumpServerAddress      134.138.62.226, 134.138.61.225;
option consoleServerAddress 134.138.61.225;
option consoleServerPort     7000;

# PXE boot
class "pxeclient" {
match if substring (option vendor-class-identifier, 0, 9) = "PXEClient";
option vendor-class-identifier "PXEClient";
}

# for the SCB's:
class "ROJ208423/1.SCB-RP" {
match if option vendor-class-identifier = "Ericsson.ROJ208423/1.SCB-RP";
filename "/tftpboot/RPI/ROJ208423-1.cfg";
}

# define the dicos processors
# wrap all the dicos processors into two shared network in order to
# get dhcpcd to answer bootp requests regarding interfaces on other
# networks (vip, external, ...).

shared-network dicosProcessor {
    subnet 134.138.61.128 netmask 255.255.255.128 {
        option routers 134.138.61.129;
        option subnet-mask 255.255.255.128;
        option broadcast-address 134.138.61.255;
        next-server 134.138.61.225;
        option telorbIpcDevice 10;
    }
    subnet 134.138.62.128 netmask 255.255.255.128 {
        option routers 134.138.62.129;
        option subnet-mask 255.255.255.128;
        option broadcast-address 134.138.62.255;
        next-server 134.138.62.225;
    }
}
```

```

option telorbIpcDevice 10;
    }
    # define virtual-ip network
}

# this is a io-processor, entry used for kick-starting the io-processor
group {

    host io1_0 {
        hardware ethernet 00:A0:A5:4C:01:75;
        fixed-address 134.138.61.225;
        filename "/opt/TSP-4.0/kickstart/ks-io-ide-eth0.cfg";
    }
    host io1_1 {
        hardware ethernet 00:A0:A5:4C:01:76;
        fixed-address 134.138.62.225;
        filename "/opt/TSP-4.0/kickstart/ks-io-ide-eth1.cfg";
    }
}

# this is a io-processor, entry used for kick-starting the io-processor
group {

    host io2_0 {
        hardware ethernet 00:A0:A5:4C:01:5E;
        fixed-address 134.138.61.226;
        filename "/opt/TSP-4.0/kickstart/ks-io-ide-eth0.cfg";
    }
    host io2_1 {
        hardware ethernet 00:A0:A5:4C:01:5F;
        fixed-address 134.138.62.226;
        filename "/opt/TSP-4.0/kickstart/ks-io-ide-eth1.cfg";
    }
}

# define dicosprocessors
group {
    host p1_0 {
        hardware ethernet 00:a0:a5:4c:01:4e;
        fixed-address 134.138.61.131;
        # this processor is a loader
        option epTftpdServerAddress 134.138.61.225;
    }

    host p1_1 {
        hardware ethernet 00:a0:a5:4c:01:4f;
        fixed-address 134.138.62.131;
        # this processor is a loader
        option epTftpdServerAddress 134.138.62.225;
    }

    host p2_0 {
        hardware ethernet 00:a0:a5:4c:01:58;
        fixed-address 134.138.61.132;
        # this processor is a loader
    }
}

```

```

        option epTftpdServerAddress 134.138.61.225;
    }

    host p2_1 {
        hardware ethernet 00:a0:a5:4c:01:59;
        fixed-address      134.138.62.132;
        # this processor is a loader
        option epTftpdServerAddress 134.138.62.225;
    }

    host p5_0 {
        hardware ethernet 00:a0:a5:50:02:3f;
        fixed-address      134.138.61.135;
    }

    host p5_1 {
        hardware ethernet 00:a0:a5:50:02:40;
        fixed-address      134.138.62.135;
    }

    host p6_0 {
        hardware ethernet 00:a0:a5:50:02:23;
        fixed-address      134.138.61.136;
    }

    host p6_1 {
        hardware ethernet 00:a0:a5:50:02:24;
        fixed-address      134.138.62.136;
    }
}

# define linuxprocessors
group {

    host p3_0 {
        hardware ethernet 00:a0:a5:4c:00:54;
        fixed-address      134.138.61.133;
        option telorbHostsData "
134.138.61.225 io1 io1net0
134.138.62.225 io1 io1net1
134.138.62.226 io2 io2net1
134.138.61.226 io2 io2net0
134.138.61.133 hn0
134.138.62.133 hn1
";
    }

    host p3_1 {
        hardware ethernet 00:a0:a5:4c:00:53;
        fixed-address      134.138.62.133;
        option telorbHostsData "
134.138.61.225 io1 io1net0
134.138.62.225 io1 io1net1
134.138.62.226 io2 io2net1
134.138.61.226 io2 io2net0

```

```

134.138.62.133 hn0
134.138.61.133 hn1
";
}

host p4_0 {
    hardware ethernet 00:a0:a5:4c:00:78;
    fixed-address      134.138.61.134;
    option telorbHostsData      "
134.138.61.225 io1 io1net0
134.138.62.225 io1 io1net1
134.138.62.226 io2 io2net1
134.138.61.226 io2 io2net0
134.138.61.134 hn0
134.138.62.134 hn1
";
}

host p4_1 {
    hardware ethernet 00:a0:a5:4c:00:79;
    fixed-address      134.138.62.134;
    option telorbHostsData      "
134.138.61.225 io1 io1net0
134.138.62.225 io1 io1net1
134.138.62.226 io2 io2net1
134.138.61.226 io2 io2net0
134.138.62.134 hn0
134.138.61.134 hn1
";
}

host p7_0 {
    hardware ethernet 00:a0:a5:50:01:b2;
    fixed-address      134.138.61.137;
    option telorbHostsData      "
134.138.61.225 io1 io1net0
134.138.62.225 io1 io1net1
134.138.62.226 io2 io2net1
134.138.61.226 io2 io2net0
134.138.61.137 hn0
134.138.62.137 hn1
";
}

host p7_1 {
    hardware ethernet 00:a0:a5:50:01:b4;
    fixed-address      134.138.62.137;
    option telorbHostsData      "
134.138.61.225 io1 io1net0
134.138.62.225 io1 io1net1
134.138.62.226 io2 io2net1
134.138.61.226 io2 io2net0
134.138.62.137 hn0
134.138.61.137 hn1
";
}

```

```

    host p8_0 {
        hardware ethernet 00:a0:a5:50:01:97;
        fixed-address      134.138.61.138;
        option telorbHostsData      "
134.138.61.225 io1 io1net0
134.138.62.225 io1 io1net1
134.138.62.226 io2 io2net1
134.138.61.226 io2 io2net0
134.138.61.138 hn0
134.138.62.138 hn1
";
    }

    host p8_1 {
        hardware ethernet 00:a0:a5:50:01:98;
        fixed-address      134.138.62.138;
        option telorbHostsData      "
134.138.61.225 io1 io1net0
134.138.62.225 io1 io1net1
134.138.62.226 io2 io2net1
134.138.61.226 io2 io2net0
134.138.62.138 hn0
134.138.61.138 hn1
";
    }
}

```

pxe.cf:

```
# ***** PXE CONFIGURATION FILE *****
# Any Line starting with a '#' is treated as a comment line
# and ignored. However, '#' must be the first character on
# a line and no spaces before that are allowed.

# ***** Telorb configuration info *****
# 1. Fill in the interface name of the interface to the otherworld,
#    in the Exclude_Network_Interface_List.
#    this is done because pxe should not serve the external nets.

# The following entry is the name of the interface on which pxe is going
# to skip.
[Exclude_Interface_List]
eth2
eth3

# This entry shows the base directory of the mtftpd. All file names/paths
# will be relative to this directory. This is the same name that should
# be used as the start up argument to the mtftpd daemon.

[Mtftpd_Root_Directory]
/tftpboot

# Set the following entry to 0 if you have a DHCP server running on this
# system.
[UseDHCPPort]
0

# Set the entry below to zero if you don't want this system to act
# as a PXE proxyDHCP
[StartProxy]
1

# Set the entry below to zero if you don't want this system to act as a
# PXE boot server
[startBootService]
1

[MasterProxy]
1

# 0 - broadcast discovery is enabled; 1 - broadcast discovery is disabled
[Discovery_BCast_Disabled]
0

# 0 - multicast discovery is enabled; 1 - multicast discovery is disabled
[Discovery_MCast_Disabled]
0

# Multicast Discovery address. The boot server would listen on this address
# for multicast boot server packets.
[Discovery_MCast_Addr]
224.0.1.2
```

```

# Prompt to display on the user screen
# format of this entry: timeout,<string>
[Prompt]
1,Press F8 to view menu ...

# This entry is set to 1 if the client should accept only responses
# from servers in this list
[Discovery_Server_List_Only]
0

# the format of the discovery_list entry is as follows
# type of the server, # of entries, Ip address of each entry
# separated by commas
[Discovery_List]
#3,2,192.215.100.49,192.215.100.99
#65535,1,192.215.100.45
#65535,2,134.138.184.201,192.168.100.100

# In some systems, the all '1' broadcast address 255.255.255.255
# won't work as the system will fail with a network unreachable message.
# In those cases, you can use the 'route add -host 255.255.255.255 eth0'
# command to add a route. Or else, you can define the host portion of the
# address alone as '1' in the following entry. This entry is not
# required if your system is set up to transmit
# broadcast packets on 255.255.255.255
[Broadcast_ip_address]
#134.138.191.255

# Architectures supported
# format : <arch-type>,<arch-string>
[PROC_ARCH]
0,X86PC

# Boot server types supported
[Service_Types]
14,Dicos
13,linux-tp
12,linux-install
3328,Dicos
0,BStrap

# Menu string that will be displayed on the client screen
# after F8 is pressed.
#
# HACK -- The second item can contain a comma separated list of ints
# within brackets, "[131, 132, 133]". If the last byte in the ip-address
# of the requesting client matches any of these, it will boot from the
# second item by default. For example; 134.138.79.131 would boot from the
# second item, but 134.138.79.135 would boot from the first for the list
# in the example above. 010924/uablrek.

[X86PC/UNDI/MENU]
14,Dicos pxe boot
13,Linux pxe boot [133, 134, 137, 138]

```

```

12,Linux install boot [225, 226]
0,Local Fot

# Image file name for DOSUNDI boot server
# format : <min layer #> <max layer #> <base file name>
[X86PC/UNDI/Dicos/ImageFile_Name]
0
1
plPXE

# Image file name for Linux-TP boot server type
# format : <min layer #> <max layer #> <base file name>
[X86PC/UNDI/linux-tp/ImageFile_Name]
0
2
linux

# Image file name for Linux install (KickStart) boot server type
# format : <min layer #> <max layer #> <base file name>
[X86PC/UNDI/linux-install/ImageFile_Name]
0
2
linux

# Image file name for BStrap boot server
# format : <min layer #> <max layer #> <base file name>
[X86PC/UNDI/BStrap/ImageFile_Name]
0
0
bstrap

# List of plug-ins. The PXEClientTester is the plug-in that
# validates received packets for PXE compliance. By default, this
# plug-in is DISABLED. Remove the '#' in front of it, if you want
# packet validation to happen
[Parsers]
# PXEClientTester
PxeParser

[Parsers/PxeParser/MainFunction]
PxeParser

[Parsers/PxeParser/DllPath]
libPxeParser.so

[Parsers/PXEClientTester/MainFunction]
PXEClientTester

[Parsers/PXEClientTester/DllPath]
libpxeTester.so

# Even if the PXEClientTester is loaded, you still need to
# turn this flag to '1' for packet validation.

[TestOn]
0

```

```
# Directory where the results of the packet validation by PXEClientTester
# will be written to. Under this directory, there will be
# sub-directories with the last 8 digits of the MAC address of
# each client.
```

```
[TestPath]
/tmp
```

```
# Option DLL for BStrap
[X86PC/UNDI/BStrap/Vendor_Dll]
libBstrapOpts.so
```

```
# Fill in a NT server name that will be contacted by the client
# to log back in for APitest
[ServerName]
#vviswana-desk2
```

```
# Specify whether the above server is in a domain or a workgroup
[IsDomain]
0
```

```
# specify the name of the workgroup or domain of the server
[Domain]
starwars
```

```
# All debug related stuff here
```

```
# If this entry is set to 1, pxe services will write
# more debug info to the system log using syslog()
[DEBUG/PxeServices_DebugOutToWindow_On]
1
```

```
# If this entry is set to 1, pxe parser will write
# more debug info to the system log using syslog()
[DEBUG/PxeParsers_DebugOutToWindow_On]
1
```

```
# If this entry is set to 1, pxe services will write
# more debug info to the file mentioned
# under the entry [DEBUG/PxeServices_Debug_FileName]
[DEBUG/PxeServices_DebugOutToFile_On]
1
```

```
# If this entry is set to 1, pxe parser will write
# more debug info to the file mentioned
# under the entry [DEBUG/PxeParsers_Debug_FileName]
[DEBUG/PxeParsers_DebugOutToFile_On]
1
```

```
[DEBUG/PxeServices_Debug_FileName]
/tmp/PxeServiceLog.txt
```

```
[DEBUG/PxeParsers_Debug_FileName]
/tmp/PxeParserLog.txt
[End]
```

Different types of start - restart

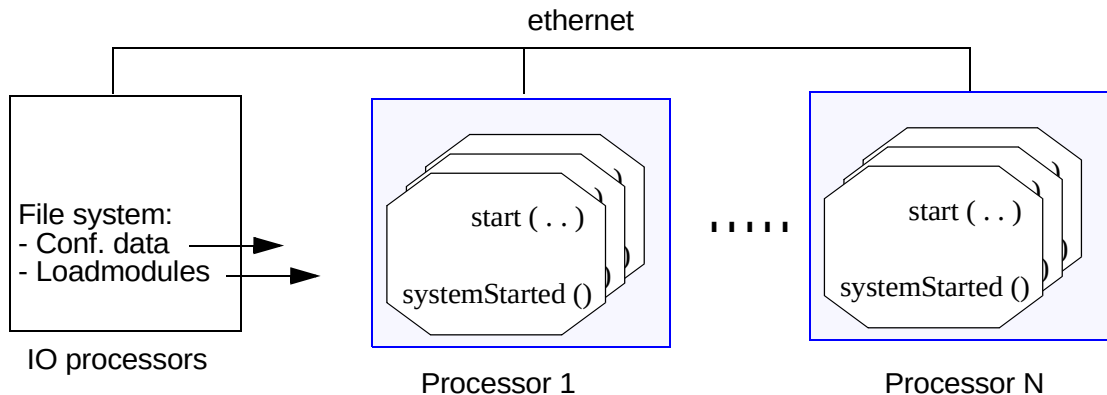
Apart from the Initial Start sequence we've just seen, there are other types of start and restart. For all different types of starts and restarts the focus is on the static processes. The dynamic processes are not involved in the start or restart procedure (although they might be affected). Each static process has a start method. It will be called with a start-reason as argument. This argument is different depending on what type of start/restart it is. The following types of start/restarts exist:

- Initial start
- Zone reload
- Processor reload
- Process restart

Initial Start

When the system is started for the first time the static processes on all processors are started. The start method will be called with start-reason= InitialStart (see fig below). When completed, the start method calls the ready method to inform the operating system. This method will be called for all static processes, on all processors. Later the systemStarted method will be called. This method must also reply by calling ready (if not the start will hang). When systemStarted has been called for all static processes, the initial start is completed. After that the system is ready to receive traffic.

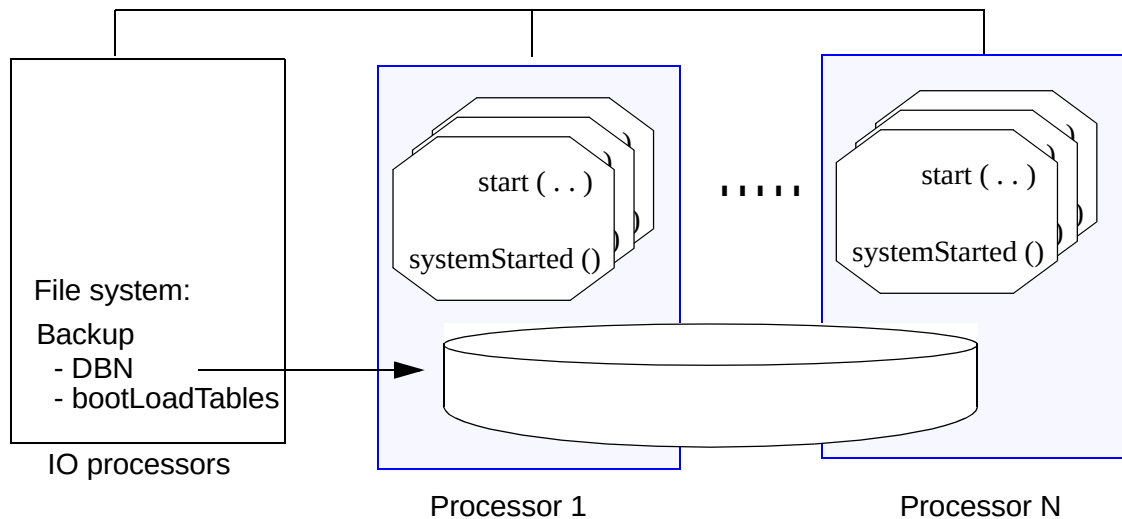
A new type of process can be added via system upgrade to an already running system. New instances of this process type will also be started with start-reason = InitialStart in the same way as described above.



Zone Reload

A zone reload implies restarting the system completely pretty much in the same way as an initial start. However the configuration and the database is restored from a previously taken backup (see figure below). There can be several backups on the IO processors but only one is “active”. A zone reload implies reverting to the state when the backup was taken.

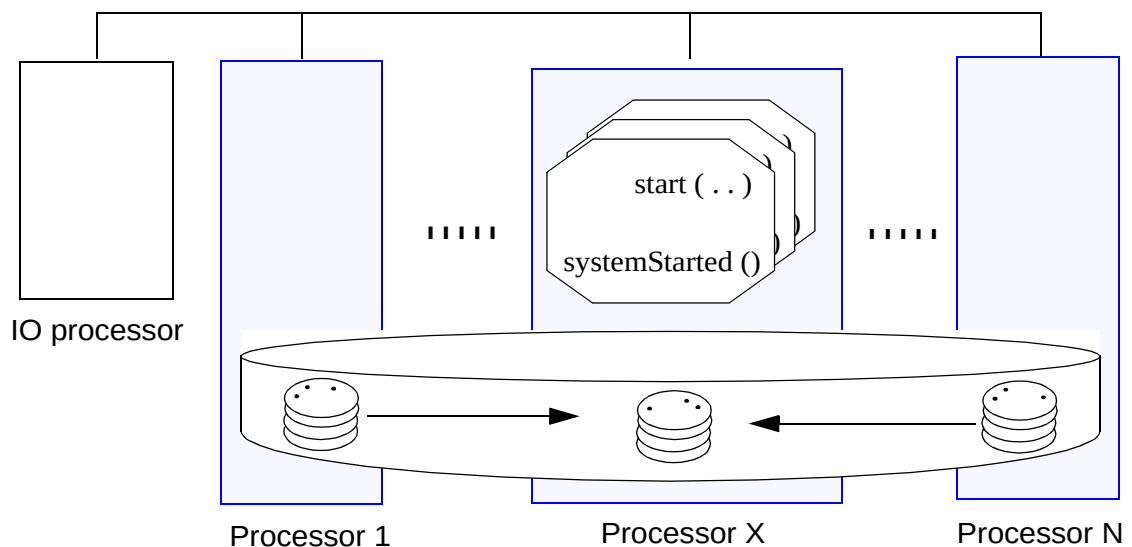
All static processes on all processors are restarted. The start method of these processes will be called with start-reason= RestartWithOldDBN. A zone reload is caused by having more then one processor unavailable at the same time. Zone reload can also be initiated through the O&M interface.



Processor Reload

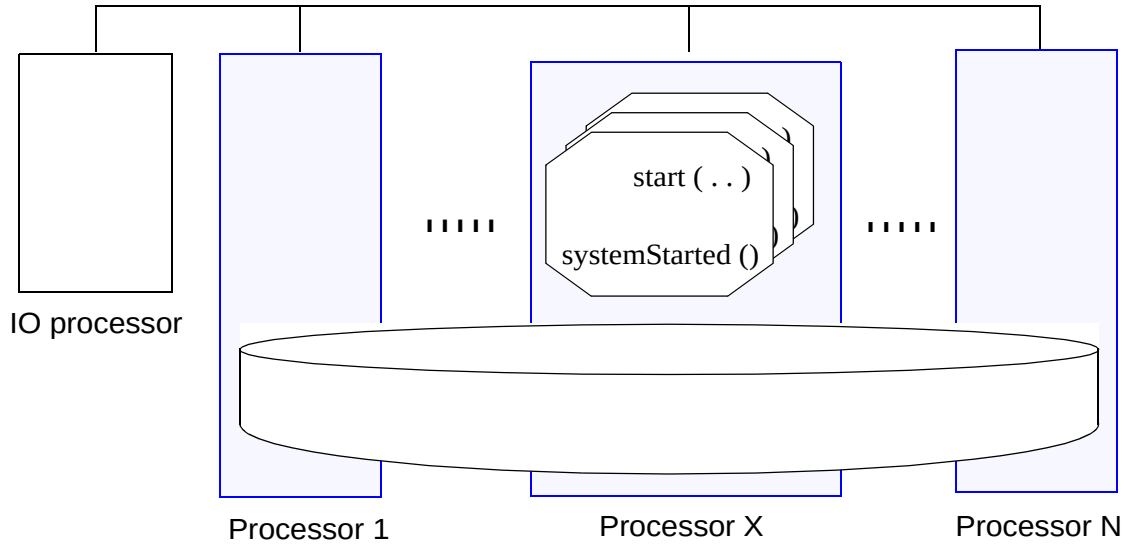
A processor reload implies restarting all static processes on one single processor. The part of the database contained in this processors is restored from redundancy replicas from other processors. The backup is not involved at all unless the processor is a loader. Then the bootloadtable is fetched from the currently active backup.

All static processes on that processor are restarted. The start method will be called with start-reason= RestartWithKeptDBN. A processor reload is caused by a kernel error (program error in the operating system itself). It can also be initiated through the O&M interface.



Process Restart

A process restart implies restarting one static processes instance. (Dynamic processes are never restarted.) The database is not changed at all. The start method will be called with start-reason= RestartWithKeptDBN. A process restart is caused by a program error in the application process.



Configuring the Traffic Processors

Now that both IOs are configured, they're ready to receive the software necessary to load the rest of the system. This software will be placed on the IOs and during Initial start it'll be loaded onto the Traffic processors.

The software to be executed on the traffic processors is delivered as a number of IDPs (Internal Delivery Packages). Each IDP contains a number of loadmodules containing the actual code.

In order to build a configuration for a particular site you must have access to:

- TSP Runtime (TSP 4 or later)
- JDK 1.3.1 for Linux

The TSP Clusterware contains the following software:

- IDP_D30-LinuxIntelPcProcessorManagement
- IDP_SU1-TelorbLinux
- IDP_D38-TelorbLinuxJava
- IDP_MD14-Dicos
- IDP_MD18-Java
- IDP_D16-FileSystemComponents
- IDP_D15-VegaFileSystem
- IDP_D40-RH7.1_Disk1
- IDP_D41-RH7.1_Disk2
- IDP_D42-RH7.1_Extras
- tools.tar

Two different modes of operation are supported.

Development Mode

In this mode, one can run "Simulated Dicos Processors" on a desktop machine. This is referred to as a 'Vega' system.

The simulated Linux Processors are called 'Mira' and can be started with the command 'eplinuxrun'.

Deployment Mode

The second mode is the deployment mode. In this mode, the TSP 4.0 clusterware and OS are run on a proper TSP 4.0 cabinet. Linux is net-booted onto some of the traffic processors, and the clusterware is automatically started. RPM loadmodules are automatically installed by the clusterware. RH7.x is used on both IO and Linux traffic processors.

TSP 4.0 Clusterware uses RPM version 4. This is included automatically in RedHat Linux 7.x, but RedHat 6.x includes RPM version 3, which is not compatible.

Starting A Deployment Mode System:

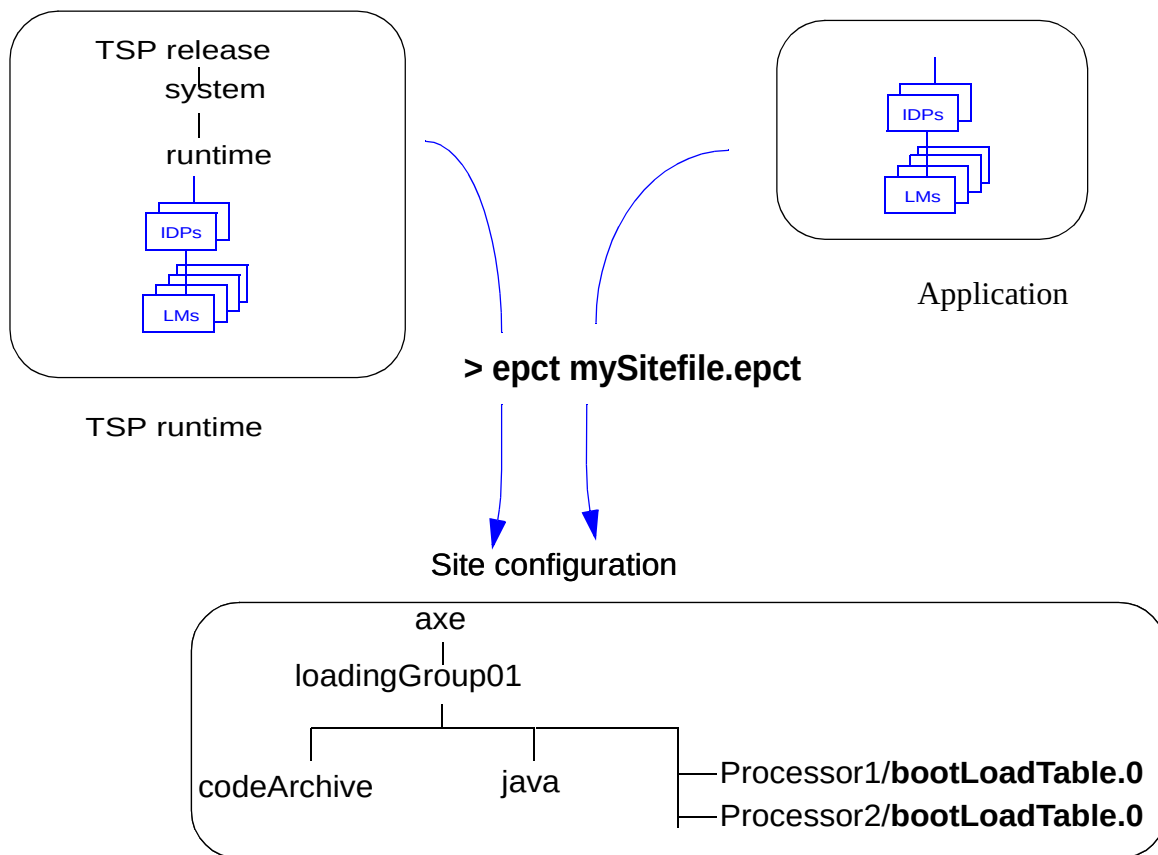
Before starting a Deployment Mode system, you must have installed the TSP4 Linux configuration onto the IO machines. This is performed according to the previous instructions:

- Install the Kickstart server.
- Edit 'telorb.conf'
- IO-Server Installation

A TSP 4.0 deployment mode runtime system must contain at least 2 TelORB-Dicos processors, and can contain 0 to n TelORB-Linux processors.

Site Configuration

The site configuration is usually built on a local workstation by the 'telorb' user and later the resulting directories will be placed on the IOs. A site configuration is built by using the epct command. One or several epct files are used as input to this command. These epct files describe the HW and SW configuration of the site.



To use the epct command and other eptools you must source the 'setup.csh' file in your TelORB distribution. This will also set a number of environment variables required by the eptools.

More information on eptools can be found at:

http://telorb.uab.ericsson.se/Doc/R14_Doc/Configuration_Parameters_and_Tools_Staging_Area/wwhelp/java/html/wwhiefs.htm

Examples of eptools are:

```
%> epct
%> epbuild
%> epconfig
%> epelfaddr
%> epgetpath
%> epfollow
```

A file can be written for ease of setting up the build environment with information about which TelORB distribution to build from:

- EnvSettings

```
#EnvSettings
set Owndir='pwd'
cd /home/dicostest/systest/R14/TelORB-R14/
source setup.csh
#source setup.csh TelORB-R14-Runtime
cd /home/dicostest/systest/R14/TelORB-R14-SDK/
source setup.csh
#source setup.csh TelORB-R14-SDK (part of TADE)
cd $Owndir
```

You should source the TSP Runtime and the TSP-SDK (in that order). Then you should epconfig the IDPs of the application,. The 'epconfig' command configures the application IDPs and adds them to the existing eptools database.

```
%> cd <path to application>
%> epconfig set -r
```

... and you should get printouts similar to this:

```
epconfig:IDP/D16
/uab/ipims_dicos/ipims_dicos10/dists/TSP4.0/Drop2.3/IDP_D16-
FileSystemComponents added
epconfig: IDP/D30
/uab/ipims_dicos/ipims_dicos10/dists/TSP4.0/Drop2.3/IDP_D30-
LinuxIntelPcProcessorManagement added
.
.
epconfig: LM/D151
/uab/ipims_dicos/ipims_dicos10/dists/TSP4.0/Drop2.3/IDP_SU1-
TelorbLinux/included/LM_D151-LinuxDicosSyslogServer added
epconfig: LM/D152
/uab/ipims_dicos/ipims_dicos10/dists/TSP4.0/Drop2.3/IDP_SU1-
TelorbLinux/included/LM_D152-LinuxProcessorSupervision added
epconfig: LM/D153
/uab/ipims_dicos/ipims_dicos10/dists/TSP4.0/Drop2.3/IDP_SU1-
TelorbLinux/included/LM_D153-TspScripts added
```

The output of the epct command is a file/directory structure containing all required code, bootloadtables etc. This site configuration will be installed on the IO processors. It is then possible to boot the TSP system.

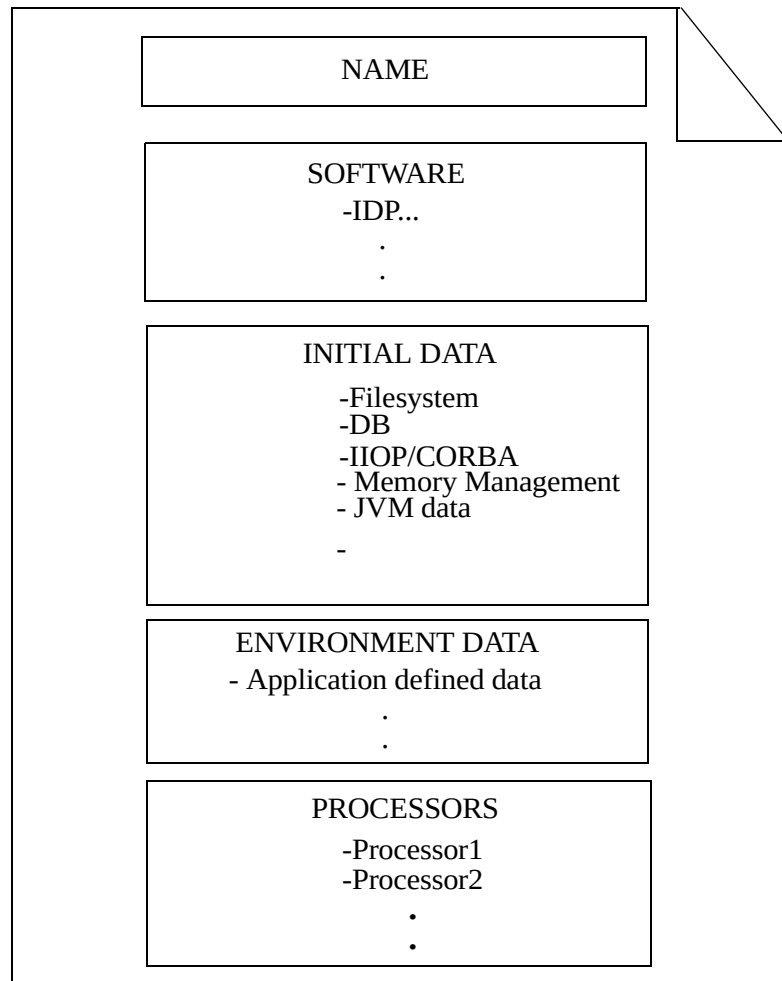
Specifying the site configuration in the epct file

Before using the epct command we must prepare the epct file. It contains:

1. the name of the site
2. software used in this site
3. initial data settings
4. “environment variables”

5. configuration data for each processor

The epct file



Details of the epct parameters may be found at:

http://telorb.uab.ericson.se/Doc/R14_Doc/Configuration_Parameters_and_Tools_Staging_Area/wwhelp/java/html/wwhiefs.htm

Epct files may either include everything in one epct file or split it up into several epct files. (e.g. name.epct, software.epct, initialdata.epct, environment.epct and processors.epct). The epct command can take multiple files as arguments.

One large epct file is used in these guidelines.

NAME

The keyword used to specify the name of the site configuration.

The name could, for instance, contain location, type of application.

Example: NAME Stockholm_UMTS_HLR_R1A;

ZONE

The keyword used to identify the system if network redundancy is being used. If there is no network redundancy the keyword ZONE is omitted.
Example: ZONE 1;

SOFTWARE

The software to be executed on a TelORB system is delivered as a number of IDPs (Internal Delivery Packages). Each IDP contains a number of loadmodules containing the actual code.

All required software must be listed in terms of IDPs under the SOFTWARE keyword in the .epct file. That includes both TelORB IDPs and application IDPs.

Required IDPs

- IDP/MD14 - The Dicos IDP is used in all systems.
- IDP/DUO2 - The Net IDP is used for extracting processor load information from the system, using the LoadViewer.
- IDP/MD16 - The DQN_SoftwareSupport_IDPEthernetIDP. It is always used in a real system.
- IDP/D13 - Include IDP/D13 if you specify one or more processors of type IntelPc under the PROCESSOR keyword (See Processors). This is the management interface for the processor type.
- IDP/D16 - Include the IDP/D16 File System Components IDP when using the new Dicos file system in a real system.
- IDP/MD11 - Include the IDP/MD11-CommLinkUdpAdaptation IDP when using Interzone IPC. It enables UDP-based IPC communication.
- IDP/MD17 - When using a simulated environment (Vega), IDP/MD16 - is replaced by IDP/MD17 (UDP). IDP/MD17 must never be used in a real system.
- IDP/D11 - Include IDP/D11 if you specify one or more processors of type Vega under the PROCESSOR keyword (See Processors). This is the management interface for the processor type. IDP/D11 must never be used in a real system.
- IDP/D15 - Include the IDP/D15 VegaFileSystem IDP when using the new Dicos file system with a Vega simulator. IDP/D15 must never be used in a real system.

Optional IDPs

- IDP/MD18 - If your application is implemented in Java you must include the Java_IDP, which contains the TelORB specific java classes.
- IDP/D2 - Include the NetRed_IDP to provide network redundancy functionality.
- IDP/D8 - Include the AXmp_IDP when using the AXmp support processor. This is the IDP for the older SS7 hardware.

- IDP/SU1 - Include the Telorb Linux IDP when using Linux processors. The Telorb systems must have both IDP/SU1 and IDP/D30.
- IDP/D30 - Include the Linux Intel Pc processor Management IDP when using Linux processors.
- IDP/D38 - Include the Telorb Linux Java IDP when using java for Linux and the TelORB Java API implementations. This IDP is optional but IDP/SU1 and IDP/D30 must be there.
- IDP/D3 - Include the KnuerrRms_IDP when using the Hardware Alarms Collector (also known as the Rack Monitoring System). A TelORB system should be configured with either:
 - GemScbSupervision_IDP (IDP/D22)
 - or, in the case of an older cabinet:
 - KnuerrRms_IDP (IDP/D3)
 - but never with both at the same time.
- IDP/D22 - Include GemScbSupervision_IDP to provide supervision for GEM cabinet hardware. (See note above)

Make sure that the last IDP entry in the list under the SOFTWARE keyword ends with a semicolon.

```
NAME Stockholm_UMTS_HLR_R1A;// can be anything
ZONE 1; // If network redundancy master

SOFTWARE
IDP/MD14, // Dicos
IDP/MD16, // DQN_SoftwareSupport_IDPEthernet
IDP/DU02, // Net_IDP, to show load on processors
IDP/MD16, // DQN_SoftwareSupport IDP
IDP/D16, // File System Components IDP
IDP/MD18, // Java, Optional
IDP/D2, // NetRed_IDP, Optional
IDP/D22, // GEM_HW_IDP, (Hardware Supervisio) Optional
.
.
.
IDP/Appl_1; // One or several IDPs belonging to application
```

Initial Data

In the INITIAL DATA section of your .epct file you can set a number of configuration parameters that are used by the lower levels of TelORB software when their default values are not suitable. Initial Data parameters cannot be changed at run-time by operators.

Initial data parameters can be divided into a number of areas:

- File System
- Database
- CORBA/IIOP
- Dumping
- Memory Management
- Java

- Loadmodule Size
- Network Redundancy
- Resource Control and Load Regulation
- PXCS (Pilot) Debugger
- Table Sizes
- Date and Time
- Other Parameters

1. FILESYSTEM

The new distributed filesystem included in TelORB-R14 must be given an initial configuration in order to be able to start up. An initial mounting of both loadingGroups and the upgradePackages directories must be specified in the epct file to be able to load the system. You specify a series of commands in the INITIAL DATA section of the epct file to create the directories and the special block device files required to mount the file system.

Commands required will differ slightly depending on file system type, NFS (Network File System) or DAS (Direct Access file System). DAS file systems are for use on Vega simulators.

Setting up a TelORB R14 file system as described here requires that the appropriate file system IDP (IDP/D16) has been specified under the SOFTWARE keyword in the .epct file.

The DUCK-server and FileServerInterpreter are no longer needed but continue to be supported for the transition period.

For a target system, this should look as follows.

First the root directory of the file system must be specified.

```
INITIAL DATA
```

```
#FileSystem
```

```
DicosFileSystemRoot=/opt/telorb/axe
```

```

FileSystemCommand_1 = "mkdir /dev"
FileSystemCommand_2 = "mkdir /loadingGroup01_1"

FileSystemCommand_3 = "mknod -b /dev/rpc1 2 1"

FileSystemCommand_4 = "mount -t nfs -o uid=101 -o gid=501 -o
ipaddr=134.138.63.25 -o ipaddr=134.138.64.25 /dev/rpc1
/loadingGroup01_1/ /opt/telorb/axe/loadingGroup01_1"

FileSystemCommand_5 = "mkdir /loadingGroup01_2"
FileSystemCommand_6 = "mknod -b /dev/rpc2 2 2"

FileSystemCommand_7 = "mount -t nfs -o uid=101 -o gid=501 -o
ipaddr=134.138.63.26 -o ipaddr=134.138.64.26 /dev/rpc2
/loadingGroup01_2/ /opt/telorb/axe/loadingGroup01_2"

FileSystemCommand_8 = "mkdir /upgradePackages"
FileSystemCommand_9 = "mknod -b /dev/rpc3 2 3"

FileSystemCommand_10 = "mount -t nfs -o uid=101 -o gid=501 -o
ipaddr=134.138.63.25 -o ipaddr=134.138.64.25 /dev/rpc3
/upgradePackages/ /opt/telorb/axe/upgradePackages"

```

Syntax can be checked at:

http://telorb.uab.ericsson.se/Doc/R14_Doc/Configuration_Parameters_and_Tools_Staging_Area/wwhelp/java/html/wwhiefs.htm

mkdir

This command makes a directory. It should be used on the root file system to create directories ready for mounting of NFS/DAS.

e.g. `mkdir /dev`

mknod

This command makes a block device.

The parameters are device_name, followed by Major number (2 for NFS/RPC, 3 for DAS/UNIX) and finally a Minor number, unique for each block device.

e.g. `mknod -b /dev/rpc1 2 1`

mount (with nfs)

This command mounts NFS mount points. Required parameters:

- o uid <user id> Set the user id
- o gid <group id> Set the group id
- o ipaddr <IP addresses of NFS servers>
- Block device to be mounted on
- Directory to be mounted on
- Directory to be mounted

```
e.g. mount -t nfs -o uid=101 -o gid=501
-o ipaddr=134.138.63.25 -o ipaddr=134.138.64.25
/dev/rpc1 /loadingGroup01_1/
/opt/telorb/axe/loadingGroup01_1
```

The userid and groupids are set to those of telorb on the IO processors. Also note that 2 IP addresses are given for each mount command, i.e.,- the 2 IP addresses of the relevant IO processor.

2. THE DATABASE

- DdbLazyPurgeInterval, indicates (in milliseconds) how often garbage collection of lazy copies takes place.
- DdbMasterMemoryShare, provides resource control of memory usage. The parameter specifies what percent of the total physical memory on each processor may be used by the database to store persistent data.
- DdbMdpTabSize, maximum number of master data partitions containing database objects.
- DdbBackupLoadmoduleSectionSize, maximum size (in bytes) of backup of one master data partition in uncompressed format. The default, 1 MB (0xF4240), is intended for simulated environments (Vega). Recommended value for target systems: 4 MB (0x3D0900).

INITIAL DATA

```
.
.
#DATABASE
DdbLazyPurgeInterval = 60000
DdbMasterMemoryShare = 70
DdbMdpTabSize = 10000
DdbBackupLoadmoduleSectionSize = 0x3D0900
```

3. CORBA/IIOP

- DorbIIOPHostName, IP address of the first and secondary IIOP-gateway. Any TelORB processor can be used as IIOP-gateway. It's recommended you set this to the ip addresses of the two loaders, using a different network for each.
- DorbIIOPPort, this is the port number to be used for running IIOP on top of TCP/IP.
- DorbIIOPBootstrapPrt, this is the port number for bootstrapping the standard CORBA Naming Service, using the new standard CORBA bootstrapping mechanism. Doesn't normally need to be set.
- CosNamingIorResponderPort, this is the TCP/IP port number at which the old proprietary bootstrapping daemon for the Corba Naming Service runs.
- CosNaming_Verbose, this enables debug output from the Corba Naming Service when debugging applications that use the Corba Naming Service.

INITIAL DATA

```
.  
.  
#CORBA/IIOP  
DorbIIOPHostName = "IP-adress1:IP-address2"  
DorbIIOPPort = 30330  
DorbIIOPBootstrapPrt = 30340  
CosNamingIorResponderPort = 7111  
CosNaming_Verbose = 1
```

4. MEMORY MANAGEMENT

HEAP_DEBUG = 1, By having this initial data setting you will get extended information from the TelORB memory management in your console printout when a process crashes due to violation of certain memory management rules. This setting is useful for example when a process is writing too far into the memory (into the next segment). The console printout will include a part telling from where in the code the nearest memory segment was allocated. The address can be translated into a certain line in the source code (by help of the `epelfaddr` command) and you will, with a very high probability, pin point the code responsible for the overwriting. Deletion of an object two times is another example of when you would get extra information.

This initial data setting should only be used in test systems because it consumes a lot more memory. N.B the extra information is not present in the error dump only the console printout.

INITIAL DATA

```
.  
.  
# Memory Management  
HEAP_DEBUG = 1
```

5. JAVA VIRTUAL MACHINE (JVM)

Modify the sizes to suit your applications.

- JAVA_HEAP - Is the size available for Java-code and Java-data. The size must be big enough to hold all your (un-packed) application classes, and all data (objects).

- JAVA_CAPSULE_HEAP

The size must be big enough to hold the JVM-code and internal data and all JavaJar-LMs that you load initially. The JavaJar-LMs are temporarily stored on the Capsule-heap until the JVM is completely started. Default value: JAVA_HEAP + 12 - Recommended value: 64 MB

```

INITIAL DATA
.
.
#Java
JAVA_HEAP = 30
JAVA_CAPSULE_HEAP = 64

```

6. LOAMODULE SIZE

- LMD_buffer_size indicates the maximum size of a loadmodule for the LM Distributor process. Default value: 3145728 byte - Recommended value for the HotSpot JVM: 8 MB.
- LMR_buffer_size indicates the maximum size of a loadmodule for the LM Receiver process. Default value: 3145728 byte - Recommended value for the HotSpot JVM: 8 MB.
- During loading, memory areas are reserved in the kernel on each processor for BOOT as BOOT_buffer_size. BOOT_buffer_size has to be as big as both the LMReceiver and LMDistributor together. Default value: 5 MB - Recommended value for the HotSpot JVM: 16 MB.

```

INITIAL DATA
.
.
#Loadmodule size
LMD_buffer_size = 8000000
LMR_buffer_size = 8000000
BOOT_buffer_size = 16000000

```

7. NETWORK REDUNDANCY

- DdbExtSynchNoOfProcesses, this is the maximum number of Ddb-DynamicDbSynchC processes on each processor that is engaged in synchronizing the database from an external backup.
- DdbnrProcBacklogBytes, this is the maximum number of bytes used for backlog in mirroring processes of each processor. When the limit is reached, the mirroring processes will start discarding the oldest transactions.

```

INITIAL DATA
.
.
#NETRED
DdbExtSynchNoOfProcesses = 4
DdbnrProcBacklogBytes = 80000000
.

```

If Network Redundancy will be used, the parameter lines below should be adjusted and added to the Initial Data section of the epct file.

```
.  
i4ddCommand_200 = "route add -net 172.18.0.0 -netmask 255.255.0.0 172.16.0.1"  
i4ddCommand_210 = "route add -net 172.19.0.0 -netmask 255.255.0.0 172.17.0.1"
```

Where 172.18.0.0 is networkA in zone 2 and 172.16.0.1 is the gateway to network A in zone 2.

Similarly, 172.19.0.0 is networkB in zone 2 and 172.17.0.1 is the gateway to network B in zone 2.

8. RESOURCE CONTROL AND LOAD REGULATION

- ProcessResourceLimit, this sets a limit to the number of system resources available to an individual process instance. System resources are resources in the operating system, such as timers, ports (on which dialogues are built), transactions, sockets, etc. The purpose of the limit is to detect and abort processes that erroneously allocate too many resources before the total number of resources in the system is exhausted.

- LoadRegulationBasicInterval, this specifies (in milliseconds) the length of the period of time between when a load is sampled and regulation parameters updated.

- LoadRegulationMax, Load Regulation is typically called by an application on entry, to determine whether there's sufficient capacity available to perform a task. If the cpu load rises above the specified load regulation level, applications which use Load Regulation will start to reject calls. There is also an operator interface to change it.

INITIAL DATA

```
.  
.
```

#RESOURCE CONTROL AND LOAD REGULATION

```
ProcessResourceLimit = 500  
LoadRegulationBasicInterval = 2000  
LoadRegulationMax = 650
```

9. PXCS (PILOT) DEBUGGER

Parameters related to Java debugging with the Pilot debugger.

- JavaPXCSDebug, this is the port number to connect to when debugging Java with the Pilot debugger. When set, the JVM is started with debugging enabled, which makes it run much slower.

```
INITIAL DATA
.
.
#PILOT DEBUGGER
JavaPXCSDebug = 30350
```

10. TABLE SIZES

- IpcPortTableSize, this is the maximum number of IPC ports on each processor.
- MonitorTableSize, this is the maximum number of simultaneous monitors on a processor. Monitors are mainly used by the JVM.
- ProcessTableSize, this is the maximum number of processes on a processor.
- SemaphoreTableSize, this is the maximum number of semaphores per processor.

```
INITIAL DATA
.
.

#TABLE SIZES
IpcPortTableSize = 32767
MonitorTableSize = 2047
ProcessTableSize = 8191
SemaphoreTableSize =2047
```

11. DATE AND TIME

- NTP_SERVER_IP, this is the IP-address of the Network Time Protocol (NTP) server. This is normally IO1. The system will try to synchronize its time with this server on initial start. This parameter is normally only used on Vega simulators, real systems get the value via DHCP. The syntax for setting this parameter is unusual, you must use commas instead of periods (dots) as separators. E.g., for an NTP server with the IP address 134.138.186.21 =NTP_SERVER_IP = (134,138,186,21)
- NTP_SERVER_PORT, this is the port for Network Time Protocol (NTP) requests. The default is 123, and it should only be altered for special tests.
- TIME_DST_DATES, this parameter is used to pre-load the Daylight Savings Time (DST) data-base at initial start. You may wish to set this parameter when initial start is used often, i.e., for Vega simulators and in a test lab environment.

E.g., In the northern hemisphere you might set the offset to 1 in March and restore it to 0 in October:

TIME_DST_DATES = (2002033102,1, 2002102702,0)

- TIME_TZ_OFFSET, set this parameter to set the local time offset in relation to UTC time. This parameter is normally only used on Vega simulators, real systems get the value via DHCP.

INITIAL DATA

.
.

#Date And Time

NTP_SERVER_IP = (134,138,186,21)

NTP_SERVER_PORT

TIME_DST_DATES = (2002033102,1, 2002102702,0)

TIME_TZ_OFFSET = 1

12. OTHER PARAMETERS

- DNS - SymbolicAddress_<n>, this is used to configure both the IP addresses of symbolic hostnames and the IP address of remote DNS servers in the TelORB DNS system.

- DNS - NumericAddress_<n>, this is used to configure the TelORB DNS to contain a symbolic hostname and its related IP address. For each value specified, there must be a corresponding 'SymbolicAddress' specification.

Values configured with SymbolicAddress_<n> and NumericAddress_<n> may be changed after the system has been initialized through the 'Config' tab of the TelorbManager application.

INITIAL DATA

.
.

#DNS

NumericAddress_51: Adjust the first IP address to correspond to the symbolic
hostname (io1), which is specified in "SymbolicAddress_51". This is the IP
address of the external interface of IO processor 1, e.g. "134.138.187.95".

NumericAddress_52: Adjust the first IP address to correspond to the symbolic
hostname (io2), which is specified in "SymbolicAddress_52". This is the IP
address of the external interface of IO processor 2, e.g. "134.138.187.96".

```

SymbolicAddress_50 = "DNS_SERVERS" # DNS for remote DNS
NumericAddress_50  = "aa1.bb1.cc1.dd1:aa2.bb2.cc2.dd2" # servers.

SymbolicAddress_51 = "io1" # DNS for IO proc. 1.
NumericAddress_51  = "134.138.187.95:172.16.100.101:172.17.100.101"

SymbolicAddress_52 = "io2" # DNS for IO proc. 2.
NumericAddress_52  = "134.138.187.96:172.16.100.105:172.17.100.105"

SymbolicAddress_53 = "localhost:localhost.localdomain" # DNS for localhost, etc.
NumericAddress_53  = "127.0.0.1"

SymbolicAddress_54 = "SCB_0A" # DNS for SCB_0A.
NumericAddress_54  = "172.16.100.100:172.17.100.100"

SymbolicAddress_55 = "SCB_0B" # DNS for SCB_0B.
NumericAddress_55  = "172.16.100.125:172.17.100.125"

SymbolicAddress_56 = "Proc_m0_s9" # DNS for TelORB proc.
NumericAddress_56  = "172.16.100.109:172.17.100.109"

SymbolicAddress_57 = "Proc_m0_s11" # DNS for TelORB proc.
NumericAddress_57  = "172.16.100.111:172.17.100.111"

SymbolicAddress_58 = "Proc_m0_s13" # DNS for TelORB proc.
NumericAddress_58  = "172.16.100.113:172.17.100.113"

```

- VIP, Currently VIP can only be used on TelORB processors running with Dicos. In this example only one VIP address is considered and the traffic processors "Proc_m0_s17" and "Proc_m0_s19" are configured for VIP usage.

If VIP will be used, the "i4ddCommand..." and "SocketUserDistributionCfg..." lines below from the template epct file should be uncommented and adjusted.

- i4ddCommand_300: This depends on your VIP router configurations. The IP addresses of the VIP router links may have to be modified, e.g. "192.168.61.1".
- i4ddCommand_310: Specify the VIP address of the zone, e.g. "100.100.100.100".

INITIAL DATA

```
.  
.
# VIP
# i4ddCommand_300 = "ddconfig i4dd0
Proc_m0_s17:eth2:192.168.61.1 Proc_m0_s17:eth3:192.168.62.1
Proc_m0_s19:eth2:192.168.61.5 Proc_m0_s19:eth3:192.168.62.5"

# i4ddCommand_310 = "ifconfig i4dd0 inet aaa.bbb.ccc.ddd"

# i4ddCommand_320 = "route delete default"

# i4ddCommand_330 = "route add default -interface i4dd0"

# SocketUserDistributionCfg_200 = "IPv4:IPPROTO_TCP:0:roundRobinDistributor"
# SocketUserDistributionCfg_210 = "IPv4:IPPROTO_UDP:0:roundRobinDistributor"
```

- KernelUtilityDaemonPort, this is the IP port number for the KernelUtilityDaemon (U-Qtil). In a “real” target system “KernelUtilityDaemon-Port” can be used when connecting to any of the processors. It doesn’t normally need to be set.

- SYSLOG_LOG_LEVEL, you can set this parameter to set the level of detail in the DicosSyslog printout.

- DicosLoadViewerServer_port, this is the (TCP) port number for DicosLoadViewer. This parameter value must match the port number given in the "-p" option to the **win** program.

INITIAL DATA

```
.  
.
KernelUtilityDaemonPort = 8100
SYSLOG_LOG_LEVEL = 5
DicosLoadViewerServer_port = 10000
```

ENDDATA

```
//#####
```

Environmental Data

Under the ENVIRONMENT DATA keyword of the .epct file you can define the applications own data. One of the main advantages of using environment data is that they can be changed at run-time by operators, whereas initial data cannot. Environment data can be accessed by the application at run-time via the DicosEnvironment interface declared in DicosEnvironment.hh for C++ applications. For Java applications this data is accessed through the se.ericsson.telorb.util.Environment class.

The data you write here will be contained in the loadingGroup01_1/ Di-cosEnvironment file of the site configuration. You can change environment data through telnet to port 8000 and going to the env directory, then use the setenv command to set an environment variable.

TelORB environment data in the .epct file can be divided into a number of areas:

- AXmp Environment Variables
- Ethernet Switch Environment Variables
- Network Redundancy Environment Variables
- RMS / HW Alarms Environment Variables
- Other Environment Variables

We look at some of the variables available in the Environmental Data section. More information can be found at:

http://telorb.uab.ericsson.se/Doc/R14_Doc/Configuration_Parameters_and_Tools_Staging_Area/wwhelp/java/html/wwhiefs.htm

1. AXmp Environment Variables

The AXmp is an external (non TelORB) support processor used for SS7 signalling. To use these environment variables, you need to have included the optional AXmp_IDP, IDP/D8.

Refer to the official documentation as above for more information.

2. Ethernet Switch Environment Variables

- ETHSW_ADDRESSES, this is a comma separated list of IP-addresses of the ethernet switches to supervise.
- ETHSW_CFG_INTERVAL, this value specifies (in milliseconds) how often TelORB checks the environment data for changes.
- ETHSW_POLL_INTERVAL, this value specifies (in milliseconds) the interval between polls to check that the switches are available.

ENVIRONMENT DATA

```
.  
.  
# Data for ETHSW supervision  
ETHSW_ADDRESSES= "192.168.1.50,192.168.1.51,192.168.2.50,192.168.2.51"  
ETHSW_CFG_INTERVAL = "1800000"  
ETHSW_POLL_INTERVAL = "15000"
```

3. Network Redundancy Environment Variables

To use these environment variables, you need to have included the optional NetRed_IDP, IDP/D2. Some of these are:

- DnrcMirrorTailDelay, this variable sets the maximum time a Smooth Takeover or a Give Back is delayed to await the tail of mirrored transactions before comparing the transactions counters.
- RSLimit, The value will be the maximum number of objects in the compensating transaction in a quick sync.
- DnrcInterZoneTimeDiff, Configurable time difference between two Network Redundancy zones. The value of this parameter should be increased if the time difference between the zones exceeds the default value of 5 minutes.
- NetRedEnableScheduler, this variable is used to turn on / off the use of Jambala Scheduler. If not set, it defaults to using the new Network Redundancy features in R14 that does not make use of the Jambala Scheduler.
- NetRedEnableZoneReloadDetection, this variable is used to enable the detection of a zone reload of the remote zone. There is a periodic verification of the state of the update channel; if this fails, the local zone perform a takeover since the other zone is not reachable, which can indicate a zone reload.

ENVIRONMENT DATA

```
.
.
# Data for Network Redundancy
RSLimit = 40000
DnrcInterZoneTimeDiff = 10
NetRedEnableScheduler = 1
NetRedEnableZoneReloadDetection = 1
```

4.GEM Cabinet Hardware Supervision

To use these environment variables, you need to have included the optional GemScbSupervision_IDP (IDP/D22).

A TelORB system should be configured with either the GemScbSupervision_IDP (IDP/D22) or, in the case of an older cabinet: KnuerrRms_IDP (IDP/D3), but never with both at the same time. For the purposes of these guidelines we use IDP/D22.

- GEM_SCB_ADDRESSES, this is a comma separated list of IP-addresses of SCBs to supervise. The SCBs should be defined with their symbolic IP names. If the GemScbSupervision_IDP is loaded but no value is specified for GEM_SCB_ADDRESSES, you will get an alarm saying:
"No GEM-SCB addresses given, HW-supervision disabled"
- GEM_SCB_POLLINTERVAL, the SCBs are polled by the TelORB system. This variable specifies the poll interval in milliseconds.
- GEM_SCB_MAXFANS, an SCB can have up to 8 fan units connected, but usually they have only one (or none). You can set the number of checked fans with this parameter.
- GEM_SCB_FANUNITS, this specifies how many fans that are connected to each SCB.

- AlarmPanelAddress, specify IO1's two IP addresses and the port on IO processor 1 for the RS232 bridge serving the alarm panel.

ENVIRONMENT DATA

```
.
.
# Data for GEM hardware supervision
GEM_SCB_ADDRESSES = "SCB_0A,SCB_0B"
GEM_SCB_POLLINTERVAL = 1000
GEM_SCB_FANUNITS = "1,0"
GEM_SCB_MAXFANS = 1

# Data for Alarm Panel
AlarmPanelAddress = "172.16.100.101;5500,172.17.100.101;5500"
```

5. Other Environment Variables

ShellHttpAccess, this variable controls access to the http interface of the TelORB Shell.

ENVIRONMENT DATA

```
.
.
ShellHttpAccess = 'OFF'
```

ENDDATA

//#####

Processors

Each processor specification begins with the PROCESSOR keyword and ends with a semicolon. For each processor the following information is listed:

- PROCESSOR - The name of the processor. This must be unique within the TelOrb system.
 - INITIAL - Used once in the file with the LOADER keyword to specify that this processor acts as initial loader.
 - TARGET - i486 | i486-linux (Intel Pentium or Intel Pentium with Linux)
 - TYPE - IntelPC | Vega | LinuxIntelPc
 - LOADER - Included if the processor acts as a loader.
- A minimum of two processors should be defined as loaders.

If your TelORB cabinet contains GEM Hardware, it's recommended that the processors you define as loaders do not all reside in the same magazine.

- ADDRESS - The ethernet addresses of the processor, two for each processor
- IN - The processor pools that the processor participates in

The keywords INITIAL and LOADER indicates that the processor plays a special role during load and start of the system.

In a real site you would also find some application defined pools. The pools found in the example below all belong to the TelORB operating system. Every pool introduced by the application must also be defined. This is done in a file called 'config.epct' stored under the IDP folder.

#Sample config.epct

```
POOL JavaDemoTGPool
POOL JavaDemoAppPool
```

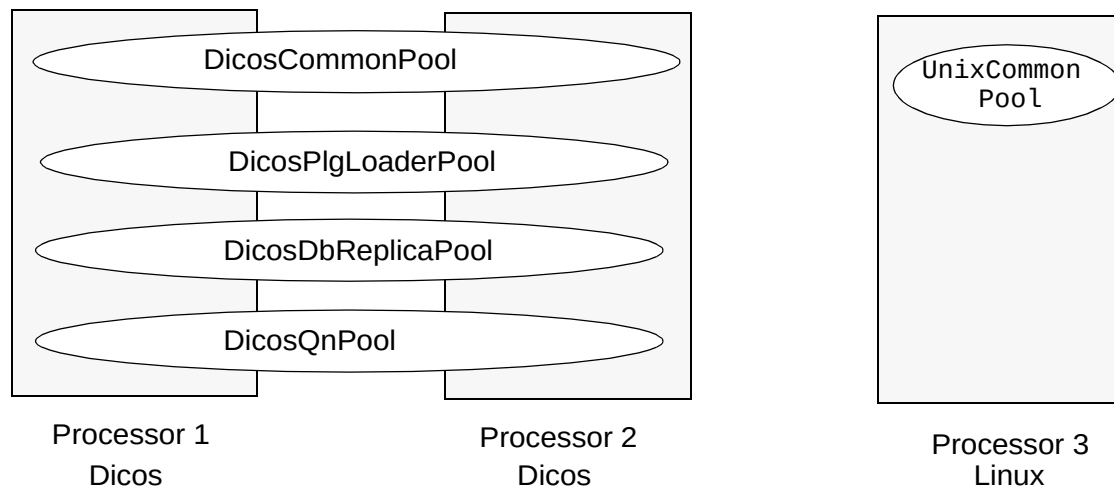
```
ALLOCATE DemoMO_Dut TO DicosPlgLoaderPool;
ALLOCATE CENTRAL_TG_Dut TO DicosPlgLoaderPool;
ALLOCATE TG_Config_Dut TO JavaDemoTGPool;
ALLOCATE CplusTG_Dut TO JavaDemoTGPool;
ALLOCATE DemoPot_Dut TO JavaDemoAppPool;
```

Processor configuration

```
PROCESSOR Processor1
    INITIAL
    TARGET i486
    TYPE IntelPc
    LOADER
    ADDRESS "00:a0:a5:4c:01:4e 00:a0:a5:4c:01:4f"
    IN DicosCommonPool,DicosPlgLoaderPool,DicosDbReplicaPool,DicosQnPool;
```

```
PROCESSOR Processor2
    TARGET i486
    TYPE IntelPc
    LOADER
    ADDRESS "00:a0:a5:4c:01:58 00:a0:a5:4c:01:59"
    IN DicosCommonPool,DicosPlgLoaderPool,DicosDbReplicaPool,DicosQnPool;
```

```
PROCESSOR Processor3
    LOADER
    TARGET i486-linux
    TYPE LinuxIntelPc
    ADDRESS "00:a0:a5:4c:00:54 00:a0:a5:4c:00:53"
    IN UnixCommonPool;
```

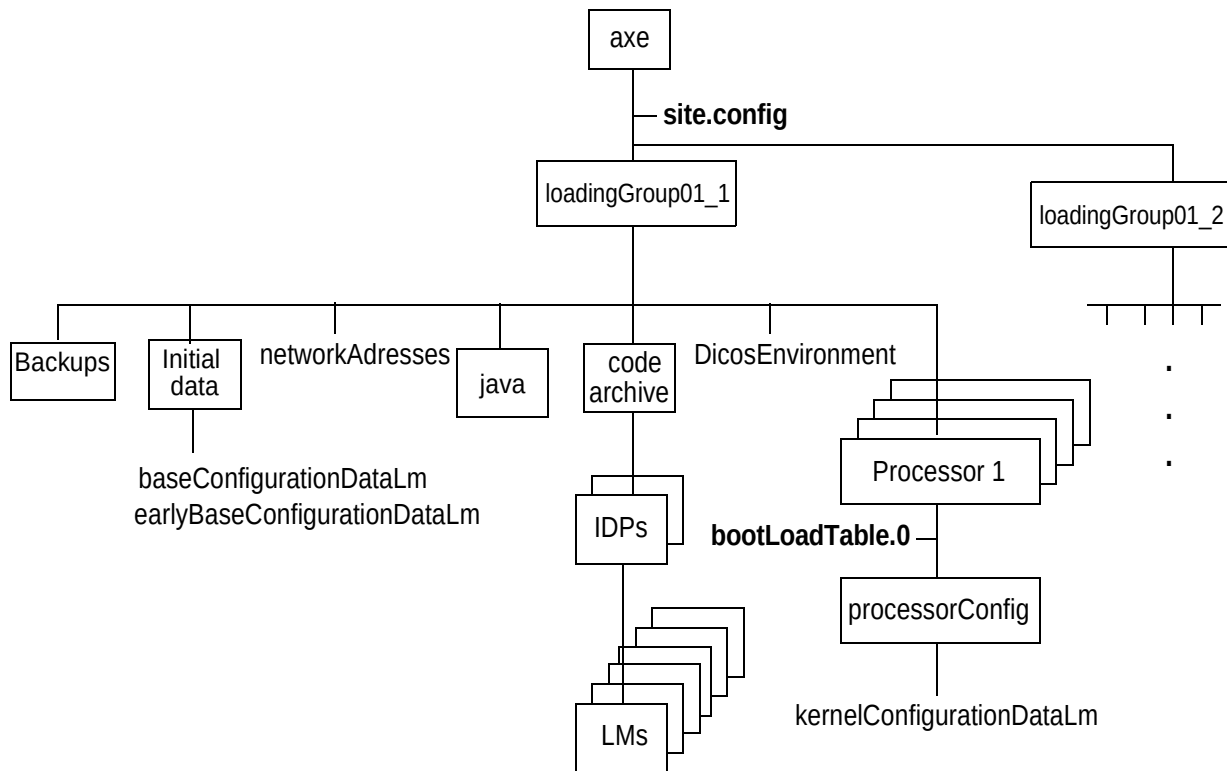


Generating the site configuration

The epct file is given as a parameter to the epct command. Any config.epct file located in the IDPs listed in the SOFTWARE section of the epct file, will be read by the command as well. (These config.epct files contains the allocation of DUTs to processor pools.) An example of an epct command, using one large epct file (site.epct), could look like this:

```
%> epct -t i486 -f loadingGroup01_1 -f loadingGroup01_2 -Fvc -d axe
site.epct
```

This command generates a directory as shown, with two loading-Groups, each one's subdirectories being identical to its peer.



Site configuration directory structure

The epct commands builds a directory structure, under an 'axe' directory. It contains the code (C++ and java) of the operating system. Apart from code there is also a lot of configuration information. The directory structure is loaded onto the IO-processors. Thereby the TelORB system can boot. The generated directory structure contains primarily:

- Processor directories:
For each processor there is a directory containing a bootload table listing what LMs are to be loaded on the procesor. These files are "the essence" of the site configuration.
- kernelConfigurationDataLm:
Contains basic data about the processor plus the INITIAL DATA section from the epct file.
- Initial data directory:
Contains two command files, baseConfigurationDataLm and early-BaseConfigurationDataLm. When these files/loadmodules are transferred to the initial loader and executed, the complete HW and SW configuration of the system, will be stored in the database locally on the initial loader. (This is the reason why we must delay the start of the second loader when starting up the system initially when we do our exercises. If the second loader is started before or at the same time it will try to act as configuration manager master but it will fail be-

cause this configuration information will be missing in the database. The database is local at this stage i.e not distributed.) At processor re-start of non-loaders the bootLoadTable can be calculated from this information in the database instead of fetching it from the IO-processor.

- network Addresses file:
Contains name and ethernet addresses of all TelORB processors (2 addresses per processor, one for each network card). Note: This file must be updated if you replace a processor with a new one.
- Java directory, contains the following Java classes:
 - the standard Java classes from jdk 1.3.1
 - TelORB's Java API and all java classes for TelORB internal use.
- codeArchive:
Contains all C++ loadmodules (TelORB)
- DicosEnvironment file:
Contains the values of the ENVIRONMENT DATA specified in the epct file.
- Backups directory:
This is empty initially, but any backups taken at a later stage in the system will be stored in here.

The generated site configuration under the 'loadingGroups' directories, will next be transferred to the IO processors of the site.

Applying the Configuration

In the previous section the epct command below was successfully run :

```
%> epct -t i486 -f loadingGroup01_1 -f loadingGroup01_2 -Fvc -d axe  
site.epct
```

In the environment where the epct command was run, an ‘axe’ directory will have been created with two sub-directories:

‘loadingGroup01_1’ and ‘loadingGroup01_2.’

This command also generates a ‘site.config’ file. This file contains all configuration data relevant to this particular epct file.

All the software is now in place for starting the TelORB system.

LoadingGroups

Currently on both IOs, as a result of the kickstart installation, there’s a ‘/opt/telorb/axe’ directory. This contains two sub-directories: ‘upgradePackages’ and ‘loadingGroup’. The ‘axe’ directory and its sub-directories were created by the ‘prepareTelORB’ script when the IOs were booted.

```
%> cd /opt/telorb/codearchive/iom_scripts  
# Extract from prepareTelORB.sh (functions.sh)  
.  
.  
mkdir -p "$OPTTELORB"/axe/loadingGroup01_1  
mkdir -p "$OPTTELORB"/axe/upgradePackages  
.  
.
```

If the IOs have just been loaded/rebooted the ‘loadingGroup’ directory on both IOs will be empty. The two ‘loadingGroups’ sub-directories that were generated by the epct command have to be copied to the live environment and will replace the empty ones. We use ‘ftp’ to transfer the directories. You need to copy ‘loadingGroup01_1’ to /opt/telorb/axe on IO1 and ‘loadingGroup01_2’ to the same place on IO2.

Before transferring the two loadingGroups you should first start up a console for each processor on the system. These consoles allow you to see what is happening on each processor, and allow you direct control of a processor, thus enabling you to stop the processor. We use the Software Terminal Server (STS) for this purpose.

STS

STS makes it possible to get log files and consoles for each of the TelORB processors. The disadvantage with STS is that it does not show the beginning of the TelORB processor loading phase. To see these early stages it's necessary to connect a terminal to a processor and use minicom to see the printouts.

STS is started as part of the 'start_services script' in /opt/telorb/etc/. The option 'consserverip' is used in 'telorb.conf' to specify the console server location (IP address of IO1 or IO2). Normally IO1 is the preferred IO processor. The consoles will be directed to the specified IO processor.

e.g. consserverip 134.138.61.225

This value is reflected in 'consoleServerAddress' in the 'dhcpd.pxe.conf' file on both IO processors.

The option 'tsport' in 'telorb.conf' is used to define the first port for the terminal-server. e.g. tsport 7000

Example of 'start_services' on IO1:

```
[telorb@to26io1 etc]$ pwd
/opt/telorb/etc
[telorb@to26io1 etc]$ more start_services
#!/bin/csh -f
source /opt/telorb/codearchive/Unix_IDP/env/setup.csh
cd /opt/telorb/codearchive/iom_scripts
if('./checkGroup.sh telorb' != "") then
    echo "To run this script you need to be member of the group telorb."
    exit
endif
setenv DUMPPDIR /opt/telorb/dumps
# Settings for ep-tftp
setenv DICOS_FILESYSTEM_ROOT /opt/telorb/axe

## Start the telorb services

./startDaemon.sh ./guardDs -th 134.138.61.131 -th 134.138.61.132 -th
134.138.61.133 -th 134.138.61.134 -th 134.138.61.137 -th 134.138.61.138 -hbt 10
-hbi 5 -rci 5 > /dev/null
./startDaemon.sh ./guardEptftp /opt/telorb/etc/ep_tftpd.conf > /dev/null
./startDaemon.sh ./guardFsi -n io1 > /dev/null
./startDaemon.sh ./guardCrashCo -i 134.138.61.225 -a 134.138.61.225 -b
134.138.61.226 -p -j -c /opt/telorb/etc/crashcollector.conf
./startDaemon.sh ./guardDump -s $DUMPPDIR -i
./startDaemon.sh ./guardSts 8 7001 -log
./startDaemon.sh ./guardSysC -L -d 5 -m io1 -t 134.138.61.131-136
./startDaemon.sh ./guardAppC -s 4M -d 8 -m io1 -t 134.138.61.131-136 HLR SCP >
/dev/null
./startDaemon.sh ./guardTrMgr /opt/telorb/etc/transfermgr.conf
./startDaemon.sh ./guardPxe -d -f /opt/telorb/etc/pxe.cf
./startDaemon.sh ./guardMirror -s 20 -i 134.138.61.226 -l 1
[telorb@to26io1 etc]$
```

Example of 'start_services' on IO2:

```
[telorb@to26io2 etc]$ pwd
/opt/telorb/etc
[telorb@to26io2 etc]$ more start_services
#!/bin/csh -f
source /opt/telorb/codearchive/Unix_IDP/env/setup.csh

cd /opt/telorb/codearchive/iom_scripts
if('./checkGroup.sh telorb' != "") then
    echo "To run this script you need to be member of the group telorb."
    exit
endif
setenv DUMPDIR /opt/telorb/dumps

# Settings for ep-tftp
setenv DICOS_FILESYSTEM_ROOT /opt/telorb/axe

## Start the telorb services

./startDaemon.sh ./guardDs -th 134.138.62.131 -th 134.138.62.132 -th
134.138.62.133 -th 134.138.62.134 -th 134.138.62.137 -th 134.138.62.138 -hbt 10
-hbi 5 -rci 5 > /dev/null
./startDaemon.sh ./guardEptftp /opt/telorb/etc/ep_tftpd.conf > /dev/null
./startDaemon.sh ./guardFsi -n io2 > /dev/null
./startDaemon.sh ./guardCrashCo -i 134.138.61.226 -a 134.138.61.225 -b
134.138.61.226 -p -j -c /opt/telorb/etc/crashcollector.conf
./startDaemon.sh ./guardDump -s $DUMPDIR -i
./startDaemon.sh ./guardSysC -L -d 5 -m io2 -t 134.138.62.131-136
./startDaemon.sh ./guardAppC -s 4M -d 8 -m io2 -t 134.138.62.131-136 HLR SCP >
/dev/null
./startDaemon.sh ./guardTrMgr /opt/telorb/etc/transfermgr.conf
./startDaemon.sh ./guardPxe -d -f /opt/telorb/etc/pxe.cf
./startDaemon.sh ./guardMirror -s 20 -i 134.138.61.225 -l 2
[telorb@to26io2 etc]$
```

The 'start_services script' of IO1 has one extra line:

./startDaemon.sh ./guardSts 8 7001 -log

Parameters:

- 8: The number of processors in the traffic system.
- 7001: The port number to be used for the first telnet setup server.
There will be as many setup servers created as the number of traffic processors in your system (8).
The first server uses port 7001 and handles output from processor 1.
The second server uses port 7002, handles output from processor 2.
The eighth server uses port 7008, handles output from processor 8.
- -log: If the parameter '-log' is entered when starting STSsw in this file, the STS places TelORB processor log files on one of the IO proc-

essors below the directory /opt/telorb/consolelogs/. The directory /opt/telorb/consolelogs/ must exist before STSsw is started.

The files are called "log1", "log1.b", "log2", "log2.b" etc. logN is continuously updated with printouts from processor N. When logN has grown bigger than the maximum file size, the file is copied "over" to logN.b and new printouts will go into a "clean" logN. The default max log file size is 2000000.

Transfer the new loadingGroups

If you have an already existing functioning TelORB system and you've generated new loadingGroups, you need to type "Ctrl-c" and "Enter" in the TelORB processor consoles to stop the TelORB processors from executing.

It's necessary to remove the old 'loadingGroups' directories from the IOs. These will be empty if you've reloaded the IOs but will contain sub-directories if using an existing system.

It's also advisable to disconnect the two loader processors from the system. This is necessary as the loaders will try and start the system as soon as they detect a 'loadingGroup' directory on either IO. It's important that both IOs contain identical 'loadingGroups' before allowing this.

The two new 'loadingGroups' are 'tarred' and then transferred to the IOs using ftp. Do not copy both subdirectories to both processors.

It's also recommended to transfer the generated 'site.config' file to the /opt/telorb/axe directory on both IOs.

Once the respective 'loadingGroup.tar' files are on their respective IOs, they can be untarred and now the system is ready to be started as an initial load.

Initial Load

The first part of the loading should always be performed by TelORB processor 1 (initial loader), so at this stage only the Initial Loader is reconnected. It is extremely important that TelORB processor 2 (the second loader) is not started / reconnected at this stage as it is not possible to perform the initial load using TelORB processor 2 as the primary loader. This is because the two command files, 'baseConfigurationDataLm' and 'earlyBaseConfigurationDataLm' from the Initial Data directory are transferred to the initial loader only. These contain the complete HW and SW configuration of the system.

The Initial Loader will detect the new 'loadingGroup' and proceed to load the other TelORB Traffic Processors.

When the TelORB processor console for TelORB processor 1 (Initial Loader) shows the line:

```
"-----> Waiting for all processors to connect to CM..."
```

... the second loader can be reconnected.

When the following printout is received in all processor consoles, the TelORB system is successfully loaded.

```
"-----> Total functionality started."
```

Loadviewer and theTelORB Manager can now be started.

Software Upgrade

Introduction

The TelORB system started in the previous section is running, but without any application. This module deals with how an application called DemoApp2000 will be loaded onto the system as an upgrade package. This application was designed to help designers, system testers and customers to test TelORB. DemoApp2000 provides an HLR-like traffic generator.

Software Upgrade

Software upgrade implies adding one or more LMs to a TelORB system. The loadmodules can be newer version of existing loadmodules or completely new loadmodules adding new functionality to the system. There are basically different ways to perform an upgrade:

- Smooth Upgrade
- Upgrade by Redundancy
- Upgrade by Reboot

Which upgrade method to use is decided automatically by the tool producing the upgrade package. The smooth upgrade method is preferable and can normally be used for applications, on the other hand the redundancy method is primarily used to upgrade the TelORB kernel and applications comprising drivers. For both types of upgrade it is required that the upgrade package containing the new loadmodules, is available in the file system of the IO processors. The execution of the system upgrade is initiated through the O&M interface of the system.

These two methods upgrade TelORB and the application without interrupting traffic. There is another method - Upgrade by Reboot - which can be used to upgrade TelORB. The Upgrade by Reboot method will cause an interruption of traffic.

Smooth Software Upgrade

First the new loadmodules will be transferred to the processor to be upgraded. For each static process instance effected by the upgrade, a corresponding new process will be started. Every process has two methods, takeOver() and concludeAndTerminate(), used during system upgrade. At one stage in the upgrade takeOver() will be called in the new process instance. The implementation of this method often leads to the setup of a so called state transfer dialogue to the old process. All information required by the new process to take over the tasks of the old process, shall

be fetched over this dialogue. The `concludeAndTerminate()` method of the old process will be executed when the state transfer is ready. Then the process is terminated.

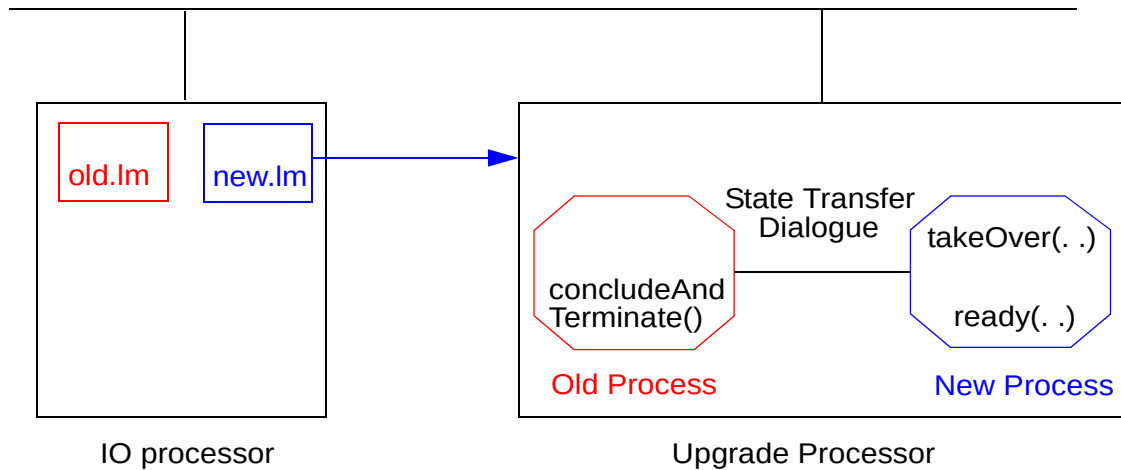


Figure 54. Smooth system upgrade

Database Upgrade

Database objects (POT's) can be upgraded with two restrictions:

- The POT must be a leaf class.
- You cannot remove attributes, only add new attributes

First all the instances of the upgraded POT are modified to include the new attribute(s). Then the `takeOver` method is called for all static processes effected by the upgrade. A database upgrade must be smooth.

Redundancy Reboot Software Upgrade

Redundancy reboot upgrade require that all software running on the processor to be upgraded is available at least on another processor. Let's exemplify with a system consisting of three TelORB processors and an IO-processor (see figure below).

To begin with the traffic-load is shared evenly between the processors. Through the O&M interface of the system, all traffic is moved from one processor at a time. The moved traffic is distributed evenly on the redundant processors. (This is done by moving distribution units in a so called planned reconfiguration.) When the upgrade processor is "empty", it is reloaded according to the new configuration including the new loadmodule. The procedure is then repeated with the next processor as upgrade processor and so on.

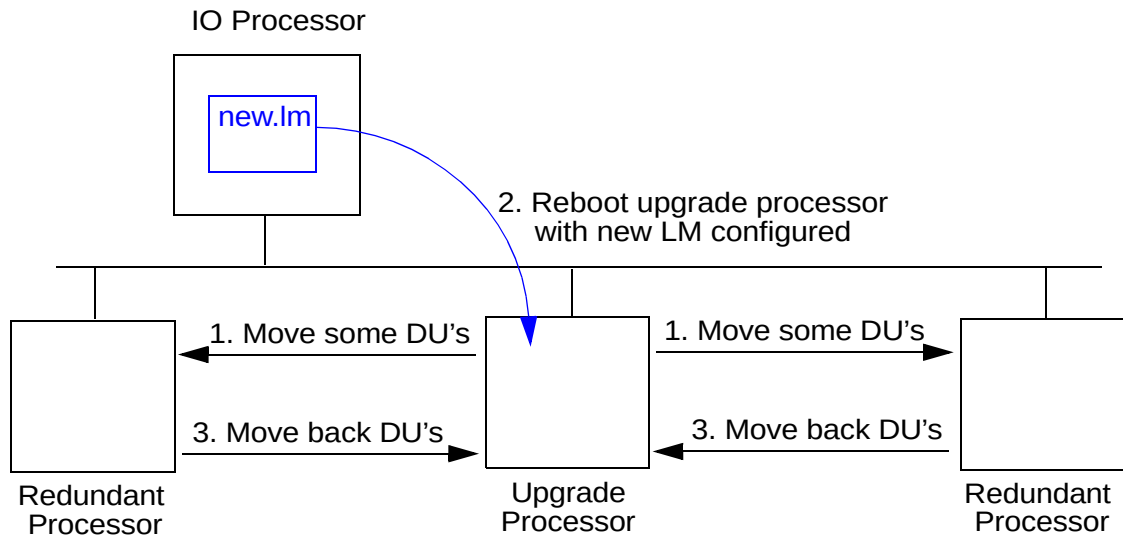


Figure 55. Redundancy reboot upgrade of one processor at a time.

Upgrade by Reboot

This method can also be used to upgrade TelORB. The Reboot method causes a reload of the entire TelORB zone, which means that the TelORB system is unavailable for the period while the reload is being performed. As a result, this method should only be used when the interruption will not affect traffic, i.e. when upgrading a system that has Network Redundancy.

The upgrade package

Upgrade packages are produced by preparing a new epct file for the site in which the name of the system must be different. Using the epct command with the -u option generates the upgrade package.

```
%> epct -t i486 -d axe -Fvc -u axe/site.config Site_for_DemoApp.epct
```

You can read more about the different options to the epct command at the TelORB web-page:

http://telorb.uab.ericsson.se/Doc/R14_Doc/Configuration_Parameters_and_Tools_Staging_Area/wwhelp/java/html/wwhiefs.htm

The generated upgrade package shall be transferred to the axe/upgrade-Packages directory of the IO-processors. (see figure below). It contains the same type of directories as the initial configuration but without the Processor directories. Another difference is the upgradeConfiguration-Data directory. It contains the upgradeControlData.01 and upgradeControlData.02 files. The information in these files will decide which up-

grade method to used (smooth or redundancy) and what “actions” that will be executed during the upgrade procedure. **N.B: It is not allowed to manually change these files or any other file in the upgrade package.**

Once the upgrade package has been transferred to the IO processors, theTelORB Manager can be used to apply the upgrade.

New persistent object types and new process types

If the upgrade package introduces new process types or persistent object types the upgrade method must be smooth. If such an upgrade packages is generated and the upgrade method choosen by the epct tool is redundancy, it won't be detected. Therefore you should always check the upgradeControlData files before carrying out the upgrading.

If the upgrade method (check in upgradeControlData.01 see below) is not “DicosSmooth” and the upgradeControlData.02 file contains new POT's or Process types, the upgrade will fail. For example the upgradeControlData.02 below implies a new POT called Sobscriber and a new PT called BotProc. The solution is to divide the upgrade into two steps (or more).

```
// Upgrade control data file 01 generated
// Thu May  2 15:39:14 2002 for uabnema
UA:mySystem3:DicosSmooth:DicosNoDbu
```

```
#####
```

```
// Upgrade control data file 02 generated
// Thu May  2 15:39:14 2002 for uabnema
IDPD:IDP/AXA860:LMA:LM/AXA880:R1A02:codeArchive/Meeting.IDP_AXA860/ApiexJava.LM
_AXA880.Code.R1A02.sparc,sparc,392240,-379247082,-2045744582
IDPD:IDP/AXA860:LMS:LM/AXA880:LM/AXA881:LM/AXG490:LM/MD32:LM/MD11
IDPD:IDP/AXA860:LMA:LM/AXA881:R1A02:codeArchive/Meeting.IDP_AXA860/ApiexC++.LM_
AXA881.Code.R1A02.sparc,sparc,47280,-214744223,-2080195790
IDPD:IDP/AXA860:LMS:LM/AXA881:LM/AXG490
IDPD:IDP/AXA860:DUT:10029,1,1:SobscriberDUT.SWI_AXA1820:SWI/AXA1820:R1A01:LM/AX
A881:10029,1,1:IDP/AXA860:Meeting:ApiexPool:0
IDPD:IDP/AXA860:POT:10028,1,1:Sobscriber.SWI_AXA1820:SWI/AXA1820:R1A01:10029,1,
1:SWI/AXA1820:SobscriberDUT.SWI_AXA1820:IDP/AXA860:Meeting:1
IDPD:IDP/AXA860:PT:10030,1,1:BotProc.SWI_AXA1520:SWI/AXA1520:R1A01:10049,1,1:SW
I/AXA820:SubscriberDUT.SWI_AXA820:IDP/AXA860:Meeting:LM/AXA880:Keyed:1:1001450
```

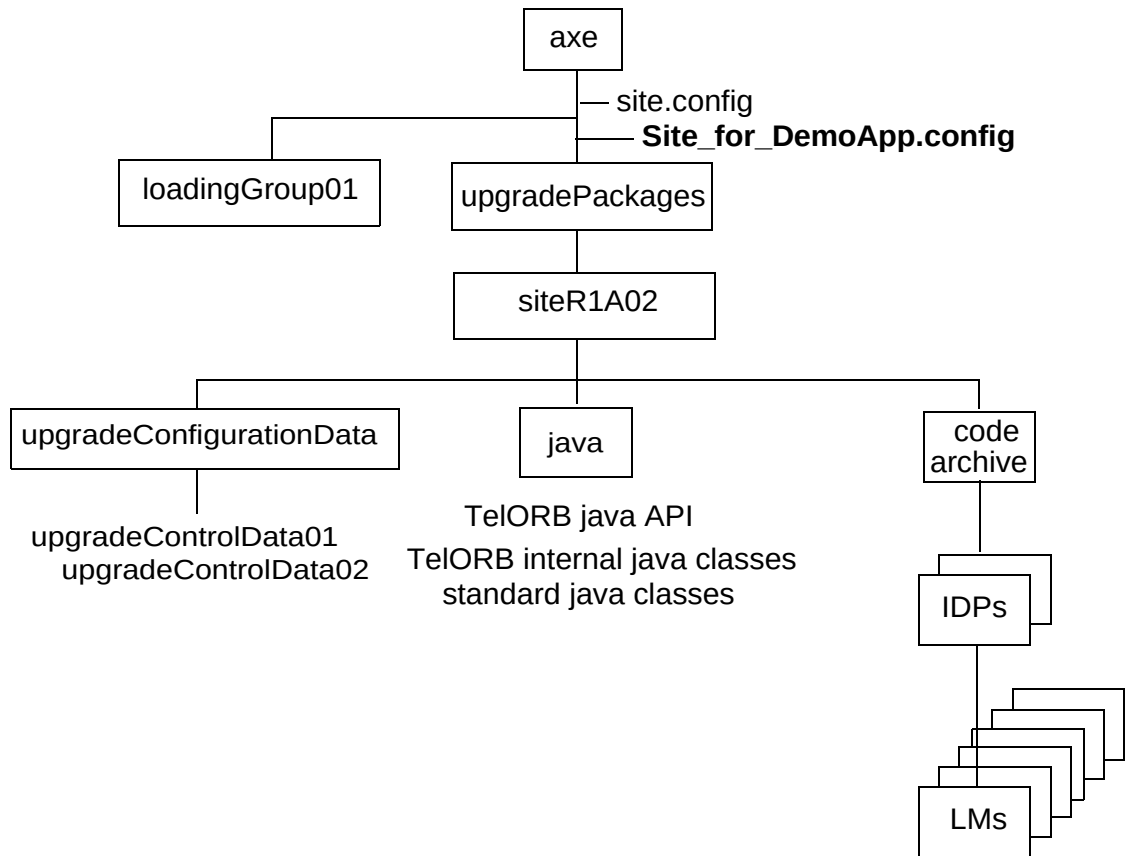


Figure 56. The configuration generated after an upgrade command

Exercises

TSP Introduction

Question 1. What Operating system runs on the IO Processors?

Question 2. List two other names for the IO Processors.

Question 3. What are the two levels of Ethernet switches within TSP 4.0?

Question 4. When would one use both levels?

Question 5. List the four standard configurations of TSP 4.0 in increasing order of size.

Question 6. When would one use both levels?

Question 7. What is an IIDP?

Question 8. What eptool is used to build a site configuration?

Question 9. What GUI would one use to configure VIP?

Question 10. What is meant by the ACID test for transactions?

Question 11. What's the basic difference between a static and dynamic process?

Question 12. What type of process should be used to ensure co-location of the process and its relevant POT?

Question 13. A fictional Midi TSP system consists of 2 IO Processors and all other Processors are used for Traffic. Four of the Traffic Processors run Linux. There are 10,000 subscribers running on the system and the DU limit is specified as being 1000. How many DUs are on each Processor?

Question 14. How many subscribers are on each Processor?

Question 15. The following are details regarding Process A, a static unique process.

Process A is distributed as DUT Sample_DUT.

.

Sample_DUT is allocated to Primary_Pool.

.

Processor1: DicosCommonPool, Primary_Pool;

Processor2: DicosCommonPool, Secondary_Pool;

Processor3: DicosCommonPool, Primary_Pool;

Upon what processor will Process A execute?

Question 16. What two methods are available to upgrade the Te-IORB-OS?

Question 17. When using normal replication, what mechanism does the DBN use to improve speed of access to POTs?

Question 18. What are the three ways of accessing DBN instances?

Question 19. There are four ways mentioned of securing Data. What are they?

Question 20. Can a POT be configured to be Disk-logged and also be configured to participate in Network Redundancy? Explain your answer.

Question 21. What's the most severe fault a TSP system can experience? What might cause this?

Question 22. How does a TSP system recover from a Zone Reload?

Configuring the IO Processors

The 'telorb.conf' file

It's necessary to write a configuration file specifying the network configuration and which processors will run linux and which will run Tel-ORB. Edit the template 'sample_telorb.conf' file to reflect the configuration of the hardware. Use the 'network_details' file to help you, this contains the network addresses and MAC addresses of the Traffic Processors.

```
%> cd <student_dir>
%> mkdir Exercise2
%> cd Exercise2
%> cp <exercise_dir>/sample_telorb.conf .
%> cp <exercise_dir>/network_details .
```

Login to IO1 and IO2 and run the following commands to retrieve network information.

```
%> ssh -l telorb <IP_Address_IO-x>
%> /sbin/ifconfig -a
%> more /etc/hosts
```

Question 23. What's the mac address of io1 on eth0?

Question 24. What's the default gateway on eth0?

Question 25. What's the mac address of io2 on eth1?

Question 26. What are the two IP addresses of the IOs for contact to the outside world?

Network Configuration

Label the following diagram according to the information found.

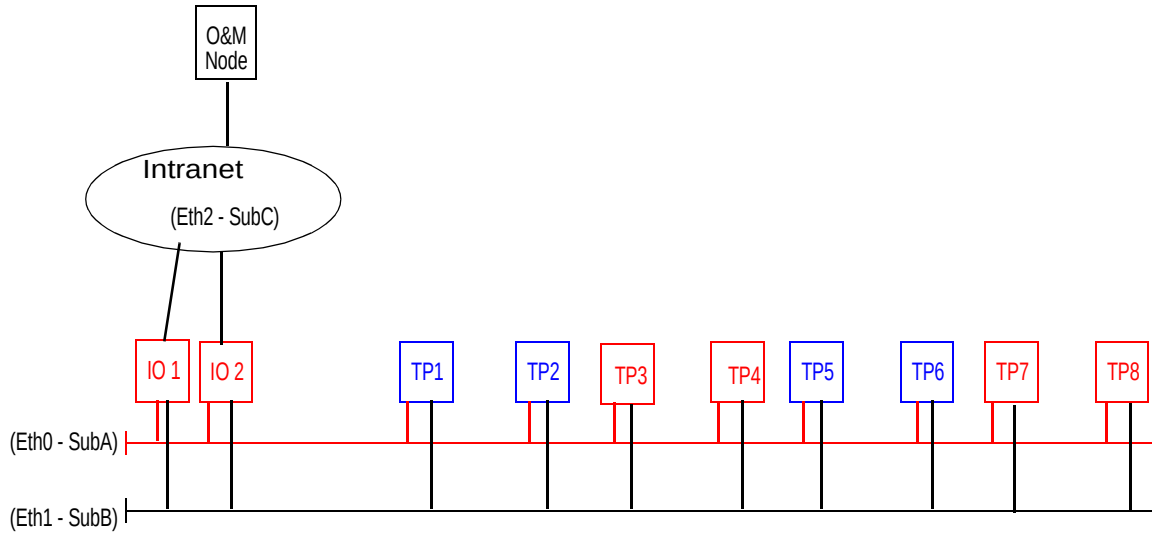


Table 1.

Address Eth0		IPAddresses P1	
Address Eth1		IPAddresses P2	
Address Eth2		IPAddresses P3	
IP Addresses IO 1		IPAddresses P4	
		IPAddresses P5	
		IPAddresses P6	
IP Addresses IO 2		IPAddresses P7	
		IPAddresses P8	
TelORB Processors		Linux Proc's	

Write the 'telorb.conf' file

Use the information obtained to edit the 'sample_telorb.conf' file and save it as 'telorb.conf'. Eth3 will not be used in these exercises and VIP and Network Redundancy will not be configured.

Use the following syntax when defining Traffic Processors. The first processor will run the TelORB OS and the second will run Linux OS.

```
dicosProc_m0_s5_0  
linuxProc_m0_s7_0
```

- m0 = magazine 0
- s5 = slot 5, the first slot available for a TrafficProcessor. Each processor uses two slots.
- 0 = ethernet 0

Use a floppy disk to copy the 'telorb.conf' file to the /opt/TSP4.0 directory of the KickStart Server. A floppy disk is usually unnecessary as the 'telorb.conf' file in this directory on the KickStart Server is normally edited directly.

Installing the IO Processors

This describes how to install the IO-processors using an external Linux KickStart server. You need the following equipment :

- A kickstart server installed according to Module 2
- A Cat5 Ethernet cable
- A 9-pin serial null-modem cable.

For this exercise there are two groups. Each group will use the KickStart Server to install one of the IOs. The same 'telorb.conf' file has to be used for both installations.

Normally the KickStart Server is only used to install IO1. IO2 is then installed from IO1. This ensures both IOs are identical. For this exercise both IOs will be installed from the KickStart Server.

- Group 1 - Install IO1
- Group 2 - Install IO2

Configuring Minicom on the Kickstart Server

Minicom is a communication program which allows the Linux kickstart server communicate with the serial communications port of an IO or Traffic processor. :

On the kickstart server open an xterm window:

```
%> su (Enter superuser mode)
%> psw (Enter password)
%> minicom -s (Enter minicom setup menu)
```

Screen and keyboard

From the configuration menu, select "Screen and keyboard" by using the arrow keys and press Enter.

Set the following parameter and press Enter.

B: DEL (followed by Enter)

This allows you to be able to enter the BIOS configuration menu, by using the "Backspace (Del) " key.

Pressing 'Esc' takes you back to the configuration menu.

Modem and dialing

From the configuration menu, select "Modem and dialing" by using the arrow keys and press Enter.

Set the following parameters (if not already set) and press Enter:

A to H: Clear all settings (Enter)(Enter)

Serial port setup

From the configuration menu, select "Serial port setup" by using the arrow keys and press Enter.

Set the following values (if not already set) and press Enter:

```
A: /dev/ttyS0      (followed by Enter)
E: 38400 8N1       (followed by Enter)
F: No              (followed by Enter)
```

Save setup as..

From the configuration menu, select "Save setup as.." by using the arrow keys and press Enter.

Enter the name 'ttyS0' and press 'Enter'.

Running minicom

To start minicom, type:

```
%> minicom ttyS0
```

Observe the 'Help' buttons. We assume it's 'Ctrl-A' but it can be other values: e.g. 'Alt'

Terminal emulation: ANSI

You can change the terminal settings by typing: 'Ctrl-A T'

Change the terminal settings to ANSI using the 'A' button and if necessary change the 'BS' key by using 'B'.

From the configuration menu, select "Save setup as ttyS0" by using the arrow keys and press Enter.

From the configuration menu, select "Exit from Minicom" by using the arrow keys and press Enter.

Verify Settings

Verify the minicom settings for ttyS0, enter the following and check the menus as described above:

```
%> minicom -s ttyS0
```

Start minicom session

Physically connect a "Null Modem" cable from the serial port on the KickStart processor with Linux OS, to the serial port on the 'target' TeIORB / IO processor.

Use an xterm - Start the minicom session:

```
%> su (Enter superuser mode)
%> psw (Enter password)
%> minicom ttyS0 (Start minicom session on previously defined ttyS0)
```

Test minicom

Connect a null modem cable from the laptop to, e.g., Processor 5 and reset Processor 5. Printout will appear in the minicom console on the laptop. Eventually the printout will redirect to the console server (on IO1). In order to see the rest of the printout, it's necessary to run an STS_console. This is part of a later exercise.

- If you need some minicom help, enter the minicom help menu:

```
%> Ctrl-a z
```

Install Linux on IO processors

When an IO is being rebooted/reloaded from the kickstart server, all existing data on the IO is wiped clear.

IO.id

Check the /opt/TSP-4.0/IO.id file on the KickStart Server, so that it corresponds with the number of the IO processor (1 or 2) upon which Linux will be installed.

```
%> more /opt/TSP-4.0/IO.id
```

If necessary, edit the file to correspond with the correct IO processor.

```
%> echo "2">/opt/TSP-4.0/IO.id
```

Verify Installation

Verify that the KickStart server will install Linux (not upgrade BIOS) by checking the /etc/pxe.cf file on the kickstart server.

```
%> more /etc/pxe.cf
```

The "Menu string..." part of the '/etc/pxe.cf' file should look as follows:

```
# Menu string that will be displayed on the client screen
# after F8 is pressed.
[X86PC/UNDI/MENU]
13,Default                                (<---- This line must be "13,Default")
13,Remote Install Linux
14,DOSUNDI
0,Local Boot
```

If "14,Default" is used, BIOS will be upgraded. If necessary, log in as superuser to edit the file.

If the 'pxe.cf' file is changed, the affected program should be restarted on the KickStart Server as follows (as superuser):

```
%> cd /etc/rc3.d/  
%> ./S56pxe stop  
%> ./S56pxe start  
%> exit
```

Stop Services

If you are reinstalling the IO-processors from scratch, make sure the DHCP- and PXE-servers are stopped on both IO-Processors before attempting an installation

Log in to both IOs as 'telorb' user, and using an xterm:

```
%> ssh -l telorb <IP_Address_IO-x>  
%> cd /opt/telorb/etc
```

Run the status scripts there, and if necessary, run the stop scripts.

```
%> ./status_services  
%> ./stop_services  
%> ./status_services  
%> ./status_dhcp  
%> ./stop_dhcp  
%> ./status_dhcp
```

If the IO processor, where Linux will be installed, is located in an existing TelORB system, halt the IO processor (from an IO processor shell):

```
%> su  
%> psw  
%> /sbin/halt
```

Connect the KickStart Server

Connect the null modem cable from the kickstart server to the serial port on the IO being installed.

Using a standard Ethernet Cable connect the kickstart server to the 100BaseTx socket on SCB-0 A(on right), this is LAN2. This involves removing the existing connection to this port on the SCB.

(This connection at the moment is going to the outside world.)

Start minicom on the laptop:

```
%> su
%> psw
%> minicom ttyS0
```

Configure the BIOS

Reboot the target IO processor by using a pin to press its reset button. Perform the next step as soon as possible after the reboot is started.

- Press the Del (backspace) key to enter the BIOS setup menu. (The current BIOS version is printed - Version 2.4.)
- Go to the "BIOS FEATURES SETUP" menu: ("Virus Protection, Boot Sequence...") by pressing 'down arrow' once and Enter.
- Select "Boot From LAN First" by pressing 'down arrow' 4 times.
- Change the BIOS boot location to "LAN 2-1" by pressing the + or - key. (It also works with "LAN 2").
- Quit the "BIOS FEATURES SETUP" menu by pressing Esc.
- Go to the "SAVE & EXIT SETUP" menu ("Save Data to CMOS & Exit SETUP") by pressing 'up arrow' 3 times and Enter.
- Answer 'y' (Yes) to the question "SAVE to CMOS and EXIT (Y/N)?" and press Enter.

Linux Installation

The Linux installation will start automatically and takes approximately 27 minutes. "Install System" should be visible in the upper right corner when the installation has started (after the booting phase). After a while, it will change to "IO-install".

Printouts appear in the minicom console, refreshing as the installation progresses.

Wait for the following installation printouts:

```
"Installation Complete"
"Complete"
"Congratulations, installation is complete."
```

Re-Configure the BIOS

Press Enter to reboot the target IO processor.

Perform the next step as soon as possible after the reboot is started, but not before "Memory Test:" is received.

- Press the Del (backspace) key to enter the BIOS setup menu.
- Go to the "BIOS FEATURES SETUP" menu ("Virus Protection, Boot Sequence...") by pressing 'down arrow' once and Enter.
- Select "Boot From LAN First" by pressing 'down arrow' 4 times.
- Change the BIOS boot location to "None" by pressing the + or - key.
- Quit the "BIOS FEATURES SETUP" menu by pressing Esc.
- Go to the "SAVE & EXIT SETUP" menu ("Save Data to CMOS & Exit SETUP") by pressing 'up arrow' 3 times and Enter.
- Answer 'y' (Yes) to the question "SAVE to CMOS and EXIT (Y/N)?" and press Enter.

The installation is completed. Let the target IO processor perform a complete reboot, to verify that the processor reboots correctly. Wait until the following printout is received:

```
"Starting crond: [ OK ]"
```

Installation Complete

This IO-processor is now installed with the following default users and passwords:

- 'root' user with a password 'rootroot'
- 'telorb' user with a password 'tel123orb'

The three CDs have loaded:

- A copy of RedHat 7.1 CD 1 & 2, plus patches.
- A DHCP server
- A PXE-server
- A TFTP-server
- A NTP-server
- A NFS-server
- TSP4 KickStart Installation Files
- Miscellaneous configuration files.

Confirm services

Verify that NFS server, DHCP server and TelORB services are running:

```
%> ssh -l telorb <IP_Address_IO-x>
%> ps -ef | grep nfs
%> cd /opt/telorb/etc
%> ./status_services
%> ./status_dhcp
```

Group2 have an extra part to do here. As the services and dhcp were manually stopped on IO1 while IO2 was being installed, it's now necessary to manually restart these processes on IO1.

```
%> cd /opt/telorb/etc
%> ./status_services
%> ./start_services
%> ./status_services
%> ./status_dhcp
%> ./start_dhcp
%> ./status_dhcp
```

For simple hardware testing, the following command can be used to test network connectivity to all processors within the node:

```
%> cd /opt/TSP4.0
%> ./ping_all
```

Ping all results:

```
[telorb@to26io1 kickstart]$ /opt/TSP-4.0/ping_all  
Reading IP from [/opt/telorb/etc/telorb.conf]
```

IO-processors:

```
134.138.61.225 [ OK ]  
134.138.61.226 [ OK ]  
134.138.62.225 [ OK ]  
134.138.62.226 [ OK ]
```

Dicos-TP-processors

```
134.138.61.131 [ OK ]  
134.138.62.131 [ OK ]  
134.138.61.132 [ OK ]  
134.138.62.132 [ OK ]  
134.138.61.135 [ OK ]  
134.138.62.135 [ OK ]  
134.138.61.136 [ OK ]  
134.138.62.136 [ OK ]
```

Linux-TP-processors

```
134.138.61.133 [ OK ]  
134.138.62.133 [ OK ]  
134.138.61.134 [ OK ]  
134.138.62.134 [ OK ]  
134.138.61.137 [ OK ]  
134.138.62.137 [ OK ]  
134.138.61.138 [ OK ]  
134.138.62.138 [ OK ]
```

SCB boards:

N/A

done.

```
[telorb@to26io1 kickstart]$
```

Run this on both IO-processors to check that all boards have booted correctly and all internal network interfaces are working.

Confirm versions

- Verify that the correct KickStart version has been installed on the IO processor and also verify the IO processor ID:

```
%> cat /etc/version
```

Example of printout on IO1(for TSP 4.0, Drop 2.3):

```
KickStart-Name: ks-io-ide-eth1.cfg  
KickStart-PN: 1/XXX 000 000  
KickStart-Rev: P1A51  
IOprocessor-ID: 1
```

- Verify the tools package version:

```
%> more /opt/telorb/codearchive/version
```

The result will depend on what version of TelORB and TSP has been sourced. A result may look like: tbAD14p8

Restore Ethernet Connection

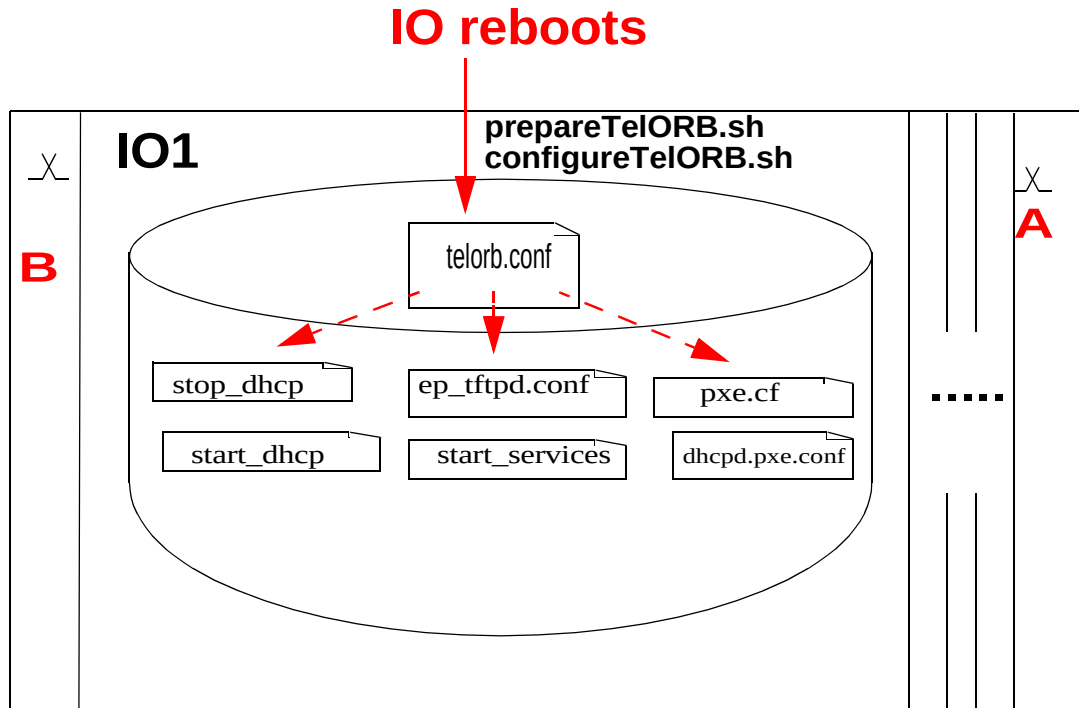
After the IO processor has been 're-installed' in the TelORB system, its connection to the outside world should be restored. Remove the ethernet connection from SCB0A to the kickstart server / laptop and reconnect the ethernet cable removed earlier from the port to SCB0A.

End the minicom session by entering:

```
%>      Ctrl-a x  
%>      Yes
```

Examining the IO Processors

Both IOs are now installed and ready to load the rest of the cluster. Before proceeding with this, log in to one of the IO processors to answer the following questions.



When the Traffic processors boot they send out a DHCP request. The dhcpd on the IOs responds using the 'dhcpd.pxe.conf' file. Examine the contents of this file.

```
%> ssh -l telorb <IP_Address_IO-x>
%> cd /opt/telorb/etc
%> more dhcpd.pxe.conf
```

Question 27. What IP addresses are defined to be dumpServers? What processors are these?

Question 28. What is the MAC address of Processor 4 on eth1?
What IP address will it be given?

Question 29. What is the MAC address of Processor 1 on eth0?
What IP address will it be given?

Question 30. If IO1 were to reboot on eth0, what file would it use
to reboot? - **ks-io-ide.cfg -??? CORRECT??????????????**

Question 31. What processor will act as Console Server?

Examine the contents of the ep_tftpd.conf file.

Question 32. What processors are not mentioned in here?
When is this important?

Examine the contents of the 'pxe.cf' file. This file is used to boot the Processors in the system.

Question 33. Which ethernet will not use pxe to boot?

Question 34. What endings are necessary in an IP address for a processor to be booted with Linux pxe boot?

Question 35. Why are there only 2 options accepted for a Linux Install boot?

Question 36. What Operating System would run on a processor with an IP address of 134.138.186.008?

Question 37. Where is the Image File found for the TelORB/Di-cos Operating System?

```
%> cd <Image_File_Directory>
```

Examine the contents of each of the 3 directories, one for each type of processor.

Question 38. How many files are loaded on a TelORB Processor?

Question 39. How many files are loaded on a Linux Processor?

Question 40. Use the command 'ls -lrt' to see the size of these files. Of the three types of processor, which has the largest amount to load? Why do you think this is so?

Examine the contents of the iom_scripts directory.

```
%> cd /opt/telorb/code_archive/iom_scripts
%> more configureTelORB.sh
```

Question 41. Observe the comments at the start of this script. According to these comments what does this script do? What other script does it call at the very beginning?

```
%> more prepareTelORB.sh
```

Question 42. Observe the comments at the start of this script. According to these comments what does this script do? What other script does it call at the very beginning?

Question 43. Both of these scripts called the same script at the very beginning. This script is in the same directory. Examine its contents. Observe the comments at the start of this script. According to these comments what does this script do?

When the IO boots from the KickStart Server, it creates an 'axe' directory. Examine the contents.

```
%> cd /opt/telorb/axe
%> ls
```

Question 44. What directories are contained in 'axe'?

Question 45. What are the contents of the 'loadingGroups' directory?

Building the SiteConfiguration

Log in to your workstation as 'telorb' user. Prepare your own site configuration by following the instructions below:

Start a new shell, preferably cmdtool.

```
%> cd <student_dir>
%> mkdir SiteConfiguration
%> cd SiteConfiguration
%> cp <exercise_dir>/mySystem.epct .
%> cp <exercise_dir>/EnvSet .
%> source EnvSet
```

Check the contents of the sourced EnvSet file!

Question 46. What is found in the directory contained in the environment variable TELORBHOME?

Question 47. What happens when you source setup.csh?

Edit the epct file according to the MAC-addresses and the IP addresses of the processors. The IIOP-gateway shall be on processor 1 and 2.

Once the file has been edited and saved you can build the site configuration:

```
%> epct -t i486 -f loadingGroup01_1 -f loadingGroup01_2 -Fvc -d axe
site.epct
```

Question 48. What is the meaning of the -F and -t options?

A complete site configuration is generated. Take a look at some of the files in this structure.

Question 49. Compare the contents of loadingGroup01_1 and loadingGroup01_2. What do you notice?

Question 50. What's contained in the 'DicosEnvironment' file?

You can investigate what's in a loadmodule by using the lduReverse command. (see example below). Search the bootLoadTables of processor 1 and processor 2.

```
%> cd axe/loadingGroup01_1
%> uabs31i1c24% lduReverse Processor1/bootLoadTable.0
%> uabs31i1c24% lduReverse Processor1/bootLoadTable.0 | grep
earlyBase
%> uabs31i1c24% lduReverse Processor2/bootLoadTable.0 | grep
earlyBase
```

Question 51. Explain the difference! When is it important?

Check the earlyBaseConfigurationDataLm and the baseConfigurationDataLm in the same way.

```
%> cd initialData
%> uabs31i1c24% lduReverse earlyBaseConfigurationDataLm
%> uabs31i1c24% lduReverse baseConfigurationDataLm
```

Checking the BIOS of Traffic Processors

It's important that the correct BIOS version (2.4) is installed on the Traffic Processors. It's also necessary that they are configured to boot from LAN-2, this ensures they will boot from the IOs.

This exercise involves using minicom to verify the BIOS version and ensuring the processors boot from LAN-2. It is done as two groups.

Run minicom

Verify the minicom settings for ttyS0 on the KickStart Server by entering the following and checking the menus as described earlier.

```
%> minicom -s ttyS0
```

Start minicom on the KickStart Server as described earlier.

```
%> minicom ttyS0
```

Verify the terminal settings (ANSI) and if necessary change them as described earlier.

Verification of BIOS

- Physically connect a "Null Modem" cable from the serial port on the KickStart Server, to the serial port on the 'target' Traffic processor, e.g., Processor 1.
- Use a pin to reset that processor. Observe the printouts in the minicom console and note the command necessary to enter the BIOS, e.g., 'Pressing F1'.
- Enter the BIOS. The BIOS version will be printed at the top of the minicom console. Verify that this is 'Version 2.4'.
- Use the arrow keys to enter the 'Boot' option. Verify that this is LAN-2.
- Press F10 to exit the BIOS setup.
- Repeat the procedure for each of the Traffic Processors.

Loading the SiteConfiguration

This exercise is done as two groups. One individual's 'axe' directory generated in the earlier exercise will be used by both Groups.

- Group1: Configure IO1 with loadingGroup01_1
- Group2: Configure IO2 with loadingGroup01_2.

To ensure the system starts with the correct configuration:

- Group1: remove Proc1 from the backplane, reconnect later.
- Group2: remove Proc2 from the backplane, reconnect later.

There are 8 Traffic Processors in the system. Only 1 console is allowed per processor. Therefore follow the instructions below to open the relevant processor consoles.

- Group1 - Consoles for Proc1, Proc3, Proc5, Proc7
- Group2 - Consoles for Proc2, Proc4, Proc6, Proc8

Start TelORB processor consoles on a WS

Each group works with their own IO and Processor Consoles. As the STS executes on IO1 only, Group2 should ensure they do not conflict with instructions being carried out on IO1 regarding the STS.

Check if STSsw is executing:

```
%> ssh -l telorb <IP_Address_IO-x>
%> ps -ef | grep STS (this should give a result on IO 1 only)
```

If STSsw is not executing, start it on IO1 by restarting the services:

```
%> cd /opt/telorb/etc/
%> ./stop_services
%> ./start_services
```

Make sure that the parameter -log is included in the STSsw command:

```
%> more start_services
```

Quit all present TelORB processor consoles, if any.

The TelORB processor consoles are started on a local WS connected to eth2. Open a shell window from a WS that is connected to eth2.

If a file that contains the startup commands does not exist, create it by using the template below or just run one or several lines from the template. Be sure to adjust <ip_io1> to valid TelORB system and IO processor.

```
----- Template for starting TelORB processor consoles -----  
#!/bin/sh
```

```
xterm -ls -sb -fn 8x13 -geometry 72x12+40+0      -sl 10000 -title "Processor1" -  
e tcsh -c 'telnet <ip_io1> 7001' &  
xterm -ls -sb -fn 8x13 -geometry 72x12+660+0     -sl 10000 -title "Processor2" -  
e tcsh -c 'telnet <ip_io1> 7002' &  
xterm -ls -sb -fn 8x13 -geometry 72x12+40+200    -sl 10000 -title "Processor3" -  
e tcsh -c 'telnet <ip_io1> 7003' &  
xterm -ls -sb -fn 8x13 -geometry 72x12+660+200   -sl 10000 -title "Processor4" -  
e tcsh -c 'telnet <ip_io1> 7004' &  
xterm -ls -sb -fn 8x13 -geometry 72x12+40+400    -sl 10000 -title "Processor5" -  
e tcsh -c 'telnet <ip_io1> 7005' &  
xterm -ls -sb -fn 8x13 -geometry 72x12+660+400   -sl 10000 -title "Processor6" -  
e tcsh -c 'telnet <ip_io1> 7006' &  
xterm -ls -sb -fn 8x13 -geometry 72x12+40+600    -sl 10000 -title "Processor7" -  
e tcsh -c 'telnet <ip_io1> 7007' &  
xterm -ls -sb -fn 8x13 -geometry 72x12+660+600   -sl 10000 -title "Processor8" -  
e tcsh -c 'telnet <ip_io1> 7008'
```

The geometry option defines the size and position of the console window (where 72x12 is the size, and +40+0 are the horizontal and vertical coordinates), and the title (as you would expect) gives a title. The <ip_io1> is the IP address of IO processor 1, and the 7001 is the port number for the software terminal server.

This software terminal server is started automatically by the start_services script, and assigns a port number to each processor, starting at 7001 and incrementing by 1 each time.

Open processor consoles

Run the file that contains the start up commands for the processor consoles, e.g. start_Group1_cons_io1 :

```
%> ./start_Group1_cons_io1
```

If the processor consoles just appear for a short while on the screen and then disappear, it's likely that another user has the consoles started elsewhere. Close the other user's consoles and retry.

If printouts do not appear on some or all TelORB processor consoles, it may be necessary to reload the affected processors.

If printouts do not appear on the processor consoles, check that the DHCP processes are executing on the IO processors:

```
%> ssh -l telorb <IP_Address_IO-x>  
%> cd /opt/telorb/etc/  
%> ./status_dhcp
```

Transfer the new loadingGroups

- Group1 handle all instructions related to IO1 and loadingGroup01_1
- Group2 handle all instructions related to IO2 and loadingGroup01_2

If you have an already existing functioning TelORB system and you've generated new loadingGroups, you need to do the following before transferring the new loadingGroups to IO1 and 2. If you've just rebooted the IOs or have never had a system, the next part - 'Existing system' - can be ignored.

Existing system

If Traffic processor consoles are already started and Traffic processors are executing:

- Type "Ctrl-c" and "Enter" in all Traffic processor consoles to stop the Traffic processors from executing.

If Traffic processor consoles are not already started:

- Start the TelORB processor consoles as described above.
- Type "Ctrl-c" and "Enter" in all Traffic processor consoles to stop the Traffic processors from executing.

It's now possible to proceed with transferring the new loadingGroups as per a new system.

New System

It's necessary to remove the old 'loadingGroups' directories from the IOs, as they'll be replaced shortly. These will be empty if you've reloaded the IOs but will contain sub-directories if using an existing system. Run the following commands on both IOs:

```
%> ssh -l telorb <IP_Address_IO-x>
%> cd /opt/telorb/axe
%> \rm -Rf loadingGroup01_1 ( loadingGroup01_2)
```

This will recursively remove the loadingGroup directories.

If not already done, gently pull out the two loader processors from the GEM racks, so that they're no longer connected in the back-plane. This is necessary as the loaders will try and start the system as soon as they detect a 'loadingGroup' directory on either IO. It's important that both IOs contain identical loadingGroups before allowing this.

In the environment where the epct was run, the 'axe' directory created contains two sub-directories:

'loadingGroup01_1' and 'loadingGroup01_2'

These loadingGroups have to be tarred and then transferred to the IOs. It's also advisable to transfer the generated '*site.config*' file (from ~/<directory_name>/axe where the system was built using the epct command) to the /opt/telorb/axe directory on both IOs.

Tar the two loadingGroups

```
%> cd ~/<directory_name>/axe
%> tar -cvf LoadingGroup_for_IO1 loadingGroup01_1
%> tar -cvf LoadingGroup_for_IO2 loadingGroup01_2
```

Do not copy both subdirectories to both processors. You must not have a directory called '/opt/telorb/axe/ loadingGroup01_1' on IO 2 or a directory called '/opt/ telorb/axe/loadingGroup01_2' on IO 1. Proceed with the following commands to place the respective LoadingGroups on their respective IO machines.

```
%> ssh -l telorb <IP_Address_IO-x>
%> cd /opt/telorb/axe
%> ftp <ip_address_local_WS>
%> pwd
%> cd ~/<directory_name>/axe
%> bin
%> get LoadingGroup_for_IO1 (LoadingGroup_for_IO2)
%> get site.config
%> bye
```

This places the 'LoadingGroup_for_01' tar file under '/opt/telorb/axe' on IO1 and the 'LoadingGroup_for_02' tar file on IO2. It places the '*site.config*' there also.

Untar and create the loadingGroup directories

```
%> tar -xvf LoadingGroup_for_IO1 (LoadingGroup_for_IO2)
```

Wait for the untarring to finish and then remove the .tar file

```
%> rm LoadingGroup_for_IO1 (LoadingGroup_for_IO2)
```

Now the system is ready to be started as an initial load.

Reload the System

Push the initial loader (TelORB processor 1) back in to position in the GEM rack. The first part of the loading should always be performed by TelORB processor 1 (initial loader). It is extremely important that TelORB processor 2 (the second loader) is not started / reconnected at this stage as it is not possible to perform the initial load using TelORB processor 2 as the primary loader.

If Traffic processor consoles are already started:

- Type "Ctrl-c" and "Enter" in all Traffic processor consoles, (except the two loaders) to stop the TelORB processors from executing.
- Type "q" in all of these consoles to restart them.

Entering 'Ctrl+C' in the consoles, and then 'q' to restart them may not work. You can also restart a processor by pressing its restart button with a pin. This may be recommended if you don't know the present state of a system.

The initial loader will detect the new 'loadingGroup' and proceed to load the other Traffic Processors.

When the TelORB processor console for TelORB processor 1 shows the line:

```
"-----> Waiting for all processors to connect to CM..."
```

..... push TelORB processor 2 (the second loader) back in to position in the GEM rack. The processors will continue to load.

Wait for all TelORB processors to initial load, i.e. until the printout

```
"-----> Total functionality started."
```

..... is received in all processor consoles.

When this printout is received in all processor consoles, the TelORB system is successfully loaded.

Examining the System

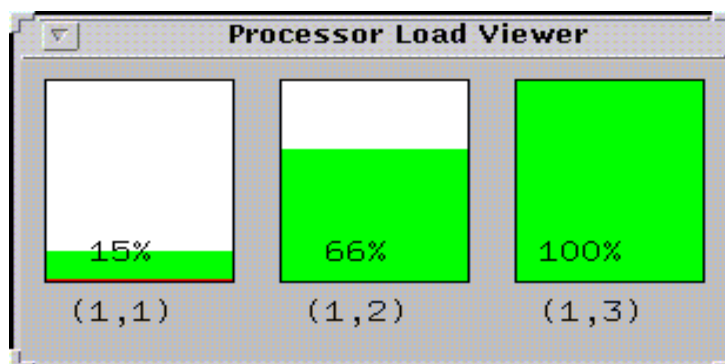
There are two GUIs which help us monitor the cluster.

- The LoadViewer allows us see a coloured, graphical representation of the load on each processor.
 - Green = normal processes
 - Red = high level processes
 - Yellow = background processes
- The TelORB Manager allows us see the configuration of the cluster and perform many maintenance operations, such as Backups, Upgrades etc.

The Loadviewer

Start the LoadViewer, open a new shell and log into an IO.

```
%> ssh -l telorb <IP_Address_IO-x>
%> pwd
%> setenv DISPLAY <myIPaddress>:0.0
%> cd /opt/telorb/codearchive/iom_scripts
%> ./win -th <IP addresses of loaders> -n -i -s &
```



Question 52. Why is there very little activity on the processors?

The TelORB Manager

The TelORB Manager is also in this directory on the IO.

```
%> ./TelORBManager -alarms &
```

The TelORB Manager will connect to one of the Processor IP addresses that were declared using the 'DorbIIOPHostname' parameter in the epct file. This GUI will be used in the remaining exercises.

Remember that the Alarms & Notifications window is an excellent mean of controlling the successful execution of the steps involved in the exercises.



Figure 57. The TelORB Manager Window

Backups and reloads will be performed in the next exercise. This is to avoid possible conflicts that may arise when performing simultaneous backups or reloads.

Question 53. What is the state of processor 1. (Click on Processors card).

Question 54. Verify that the appropriate Operating Systems are on the relevant Processors.

Question 55. How many LMs are on Processor 4?

Loading an application as a Software Upgrade

Presently there's a TSP system running but there's no application. This exercise loads a Traffic Simulator application, DemoApp2000, as a Software Upgrade.

TelORB Traffic Simulator

This section deals with an application designed to help designers, system testers and customers to test TelORB. This application is called DemoApp 2000 and provides an HLR-like traffic generator.

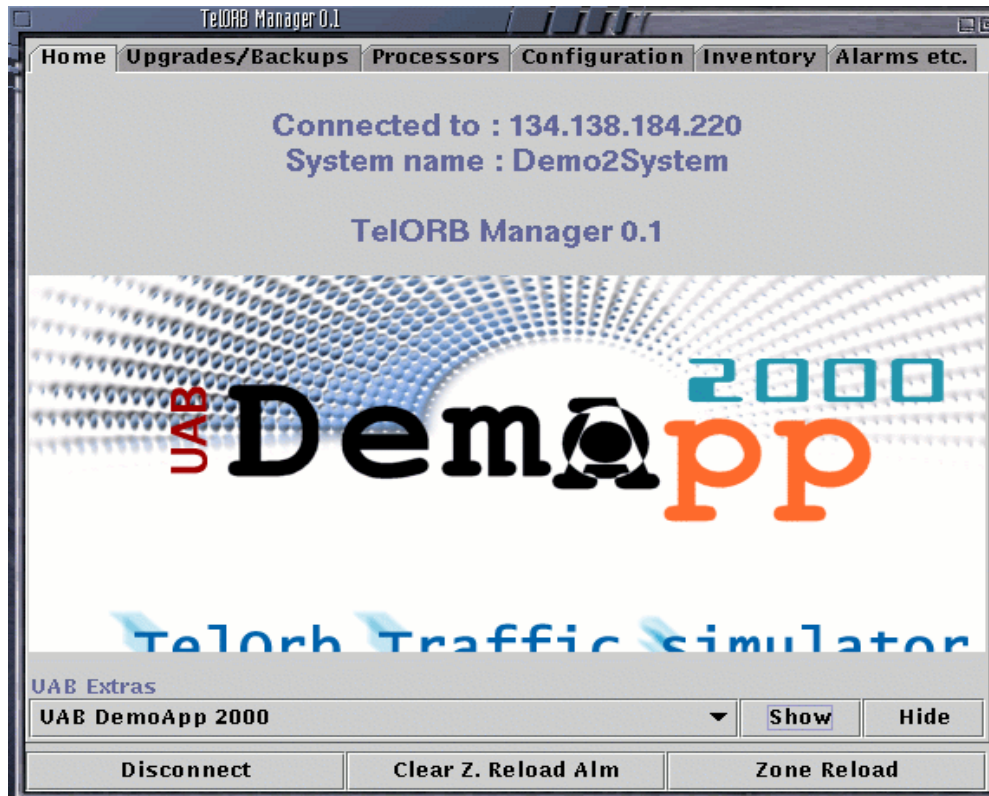


Figure 58. The TelORB Manager window.

Overview

This DemoApp2000 application contains several processes, see figure 59, starting from the top we have the De2Co process which acts like a “coordinator”, communicating with the TelORB manager via CORBA calls.

The coordinator in turn communicates with the traffic generator, De2TG, and the “populator”, De2Pop via dialogues. Traffic generator generates the traffic according to the parameters set in the DemoApp2000 window, see figure 60, the populator created the subscribers or destroys them, see the POPULATE/DEPOPULATE buttons, before you run the traffic generator the database has to be populated.

We also have two POTs, De2CoPot and Subscriber, the first one is used by the “coordinator” process, it stores information about population ranges and other application variables. The Subscriber POT represent the POT on which the subscriber is implemented.

Table 2. Persistent Object Types on the Demo2000

<i>POT</i>	<i>DISTRIBUTED AS</i>	<i>FUNCTION</i>
De2CoPOT	De2CoPOT_DUT	Stores info about the application itself
Subscriber	De2SubDUT	The subscriber implementation

Table 3. Processes in the Demo2000 application

<i>Process Name</i>	<i>Properties</i>	<i>Distributed As</i>	<i>Function</i>
De2DisPT	Central Static Replicated	De2DisDUT	Distributor
De2TGPT	Central Static Replicated	De2TGDUT	Traffic Generator
De2PopPT	Central Static Replicated	De2PopDUT	Populator/Depopulator
De2CoPT	Central Static Unique	De2CoDUT	Coordinator
De2SubPT	Key-Identified Dynamic Unique	De2SubDUT	Used to access the Subscriber POT

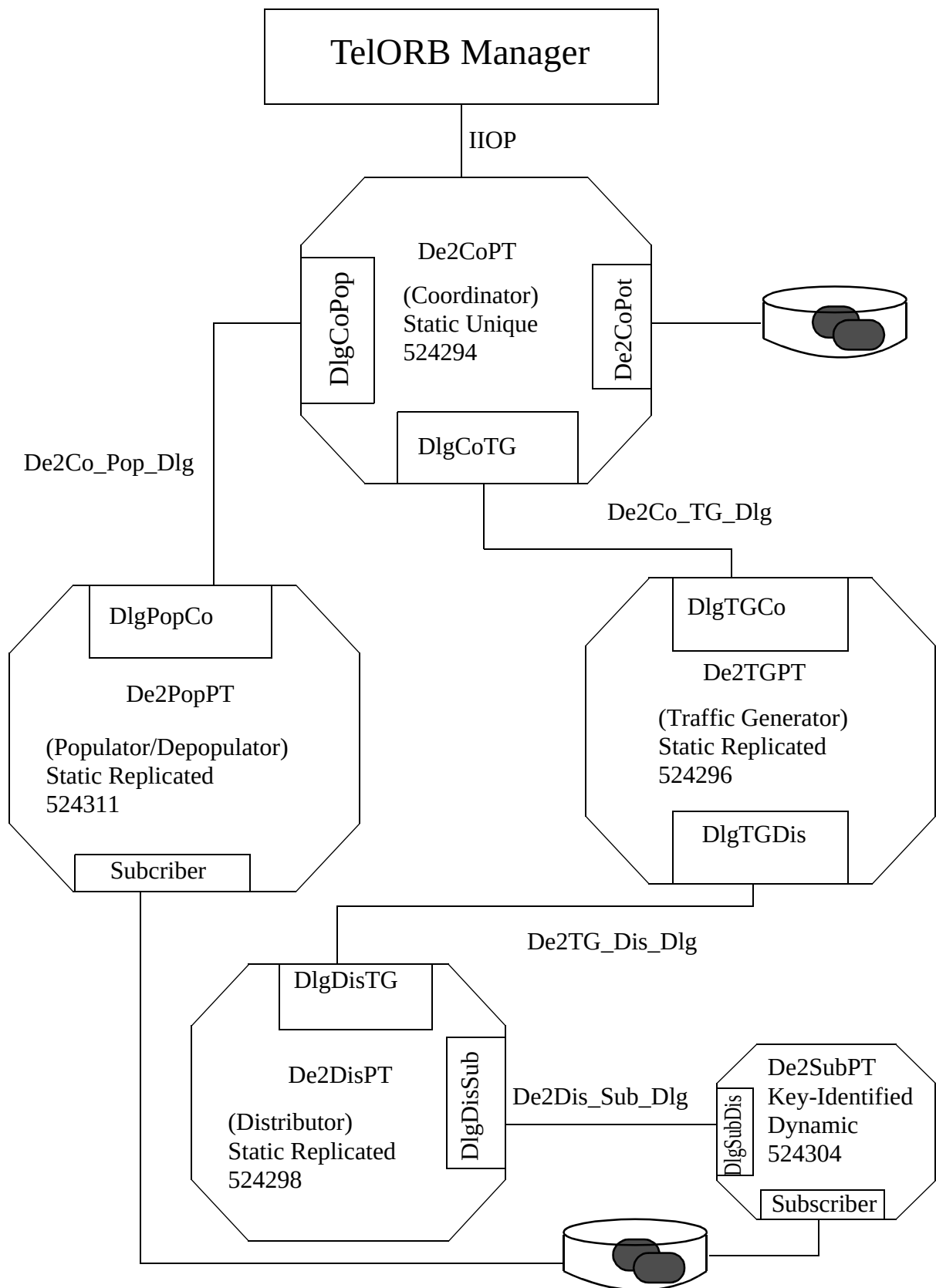


Figure 59. Processes, Dialogues and POTS in the Demo200 application

To complete the description of the Demo2000 application we need to fill in the following table where each DUT is allocated to a Pool, we ask you to complete this table, according to the config.epct file.

Table 4. DUTs and Pools

<i>DUT</i>	<i>Pool</i>

DemoApp2000 consists of two sides, the traffic generator itself which is loaded on TelORB under IDP/DEM101, and the client side contained in an applet called TelORB Manager, see figure 58.

Select the 'Home' tab in the TelORB Manager and UABDemoApp2000 is chosen on the slide menu, lower part of the window. The show button must be pushed for the DemoApp 2000 to be shown.

If DemoApp2000 is not shown, it is because the DemoApp2000 is not running on the given host, if this is not the case you will get the DemoApp 2000 window, see figure 60.

Update(%)	34	Read(%)	33
Create/Delete(%)	33	OP4(%)	0
Intensity	15	calls/sec	☒ Load Control
Print interval	2	Seconds	Type of trace
Size of Objects	4	Bytes	No trace
Ranging From	0	Ranging To	10

POPULATE	DEPOPULATE	Range Settings
RUN	STOP	Check Statistics
JRUN	JSTOP	Hide this window
Enable NPR	Load Ranges	

Figure 60. The DemoApp 2000 client side.

Database Operations

We can perform 4 different operations: Update, Read, Create/Delete and OP4, which is not implemented at the moment. We can decide which type of traffic is wanted just filling the percentage field for each operation, in the example 34% Update, 33% Create/Delete and 33% Read.

Intensity

This field specifies the number of calls per second.

Load Control

The load control checkbox lets the user to choose if TelORB control over traffic is wanted or not, if it is then TelORB will decide if a given database operation will be done depending on the processor load.

Print interval

This field allows us to choose the time interval for printouts from the TelORB, but if the interval is set to 0 then we'll get printouts after every 1000 calls.

Size of Objects

With this field the size of subscribers created is set.

Type of Trace

With this sliding menu you can choose if you want to trace different operations. The only operation that can be traced so far is pop/depop of subscribers.

Ranging from and Ranging To

The population/depopulation range or the range on which the traffic is running is specified here.

Populate

Populate lets the user create subscribers, the parameters sent to the host from the client are the following:

- Ranging From: shows where the range is started.
- Ranging To: shows where the range ends.
- Size of Objects: Size of the subscribers being created.
- Type of Trace: if the Populating should be traced.

Depopulate

Depopulate is for removing all or some of subscribers, the parameters sent to the host by Depopulate are:

- Ranging From: shows where the range is started.
- Ranging To: shows where the range ends.
- Type of Trace: if the Depopulating should be traced.

Range Settings

When this button is pushed then a window with information about populated ranges is shown, if the windows is already shown then it will be updated.



Range number	Ranging from	Ranging to	Object size(bytes)
1	0	1000	5
2	1001	2000	6
3	2001	3000	7
4	3001	3675	34

Figure 61. The Range Settings Window

Run/Stops

Starts and stops the traffic

Check Statistics

When clicking this button you get a printout which represent the sum of all printouts given by all the traffic generators running in the system.

NPR

This is used to enable/disable Netshared Partition Redundancy (Network Redundancy). Before you start populating or depop check/change the system from/to Netshared Partition Redundancy.

Printouts

The traffic generator process, De2TGPT, is central static replicated, it means that for each processor in the pool De2TGPool a De2TGPT will be started and the following printouts will be displayed, where “Trafficgenerator number” identifies the traffic generator instance, see figure 62.

```
DEBUG INFO: Trafficgenerator number:          1
DEBUG INFO: Time to generate last 1000 calls:    5130 ms
DEBUG INFO: Total number of generated calls:    47000
DEBUG INFO: Total number of aborted calls:      0
DEBUG INFO: Total number of timedout replies:   0
DEBUG INFO: Calls canceled due to loadcontrol:  0
DEBUG INFO: Requested intensity per trafficgenerator: 200 calls/sec
DEBUG INFO: Given intensity per trafficgenerator: 200 calls/sec
DEBUG INFO: This trafficgenerators actual intensity: 194 calls/sec

DEBUG INFO: Trafficgenerator number:          2
DEBUG INFO: Time to generate last 1000 calls:    46985 ms
DEBUG INFO: Total number of generated calls:    10000
DEBUG INFO: Total number of aborted calls:      0
DEBUG INFO: Total number of timedout replies:   0
DEBUG INFO: Calls canceled due to loadcontrol:  0
DEBUG INFO: Requested intensity per trafficgenerator: 200 calls/sec
DEBUG INFO: Given intensity per trafficgenerator: 200 calls/sec
DEBUG INFO: This trafficgenerators actual intensity: 21 calls/sec
```

Figure 62. Traffic generator printouts

After Clicking on check Statistics you will get:

```
DEBUG INFO: Total number of generated calls:    24477
DEBUG INFO: Total number of aborted calls:      0
DEBUG INFO: Total number of timed out replies:  0
```

Figure 37 shows the file structure of the Demo2000 IDP, the file Demo2000 SE is shown on figure 64.

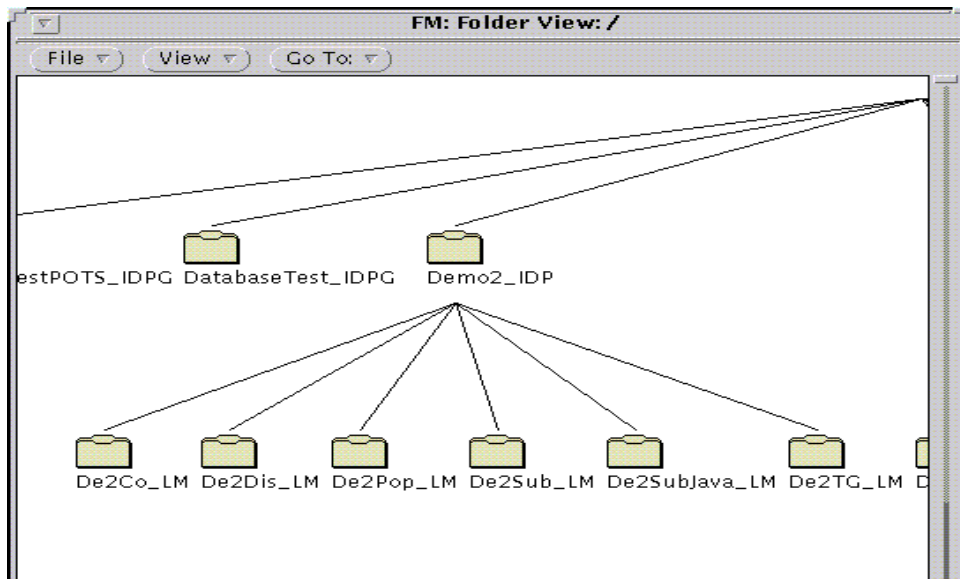


Figure 63. The Demo2000 IDP, folder view.

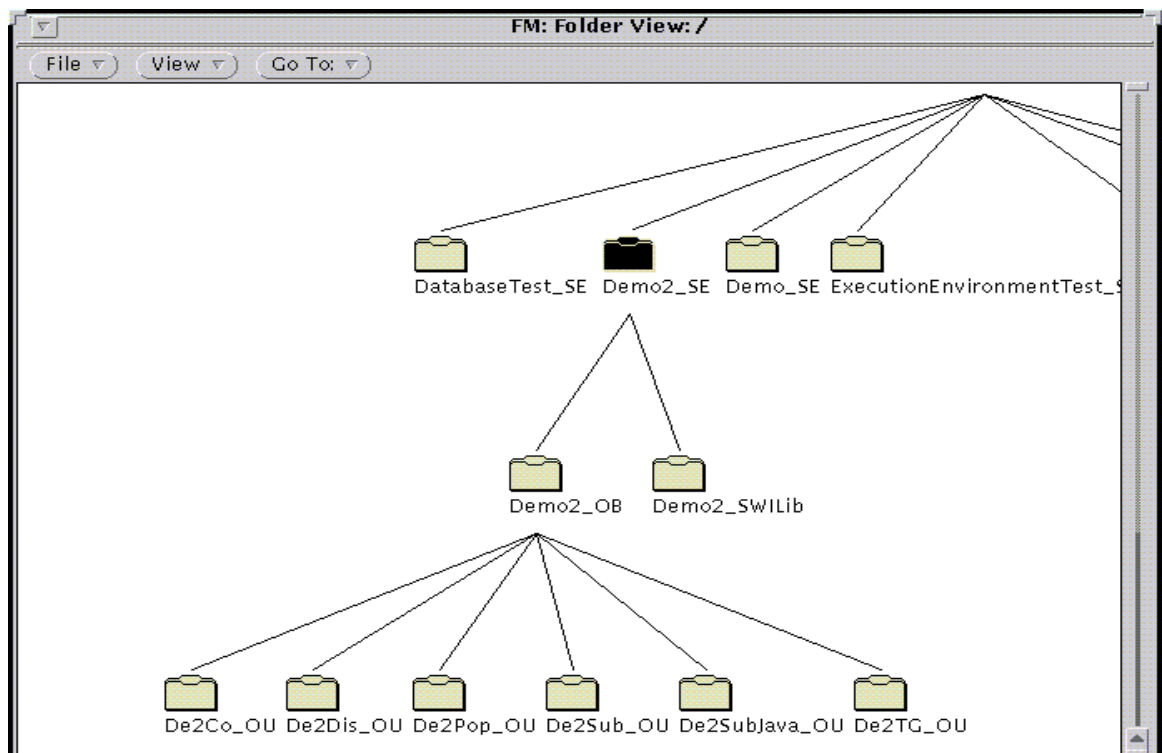
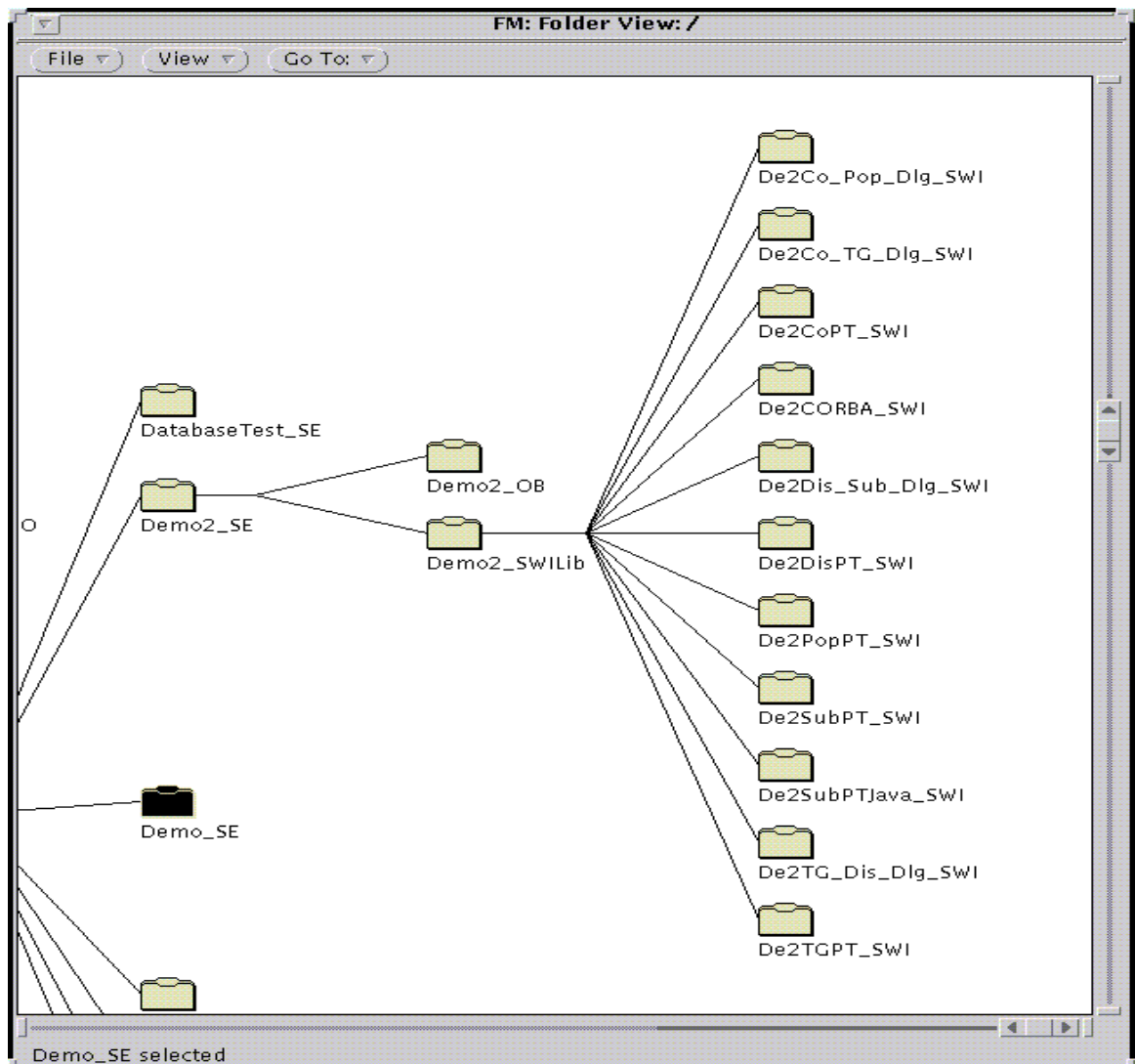


Figure 64. Demo2000 Object Units



New .epct files

In this exercise:

- Each of those in Group1 in earlier exercises will write a 'Site_for_DemoApp_G1.epct' file and then transfer a generated up-Gradepackage and the related .config file to both IOs.
- Each of those in Group2 in earlier exercises will write a 'Site_for_DemoApp_G2.epct' file and then transfer a generated up-Gradepackage and the related .config file to both IOs.

The only difference between both epct files will be the Pool allocation on the various processors.

Write Site_for_DemoApp_Gx.epct

Prepare a 'Site_for_DemoApp_Gx.epct' file, where x is your Group number. In this new file the first line should be changed to:

"NAME Site_for_DemoAppGx"

There are 9 pools defined in this application. These need to be added to the processors. Some have to be on specific processors, others may be placed on a selection of processors.

- De2CoordPool - should be placed on the 2 loaders
- De2GwPool - should be placed on the 2 loaders
- De2SubNonCoLocatedPool - should be placed on the Linux Processors
- De2TGPool, - can be placed on some or all of the other TelORB Processors
- De2DisPool, - can be placed on some or all of the other TelORB Processors
- De2UnixAccessPool,- can be placed on some or all of the other TelORB Processors
- De2SubPool, -can be placed on some or all of the other TelORB Processors
- De2PopPool, - can be placed on some or all of the other TelORB Processors
- DicosUDCRemotePool; - can be placed on some or all of the other TelORB Processors

Build an upgrade package

Logged in as 'telorb' user on your local WS, use the following command to build an upgrade package.

```
%> epct -t i486 -d axe -Fvc -u axe/site.config Site_for_DemoApp_Gx.epct
```

- Check the file structure of the generated upgrade package, compare to figure 65.

Question 56. What information do you find in the upgradeControlData files?

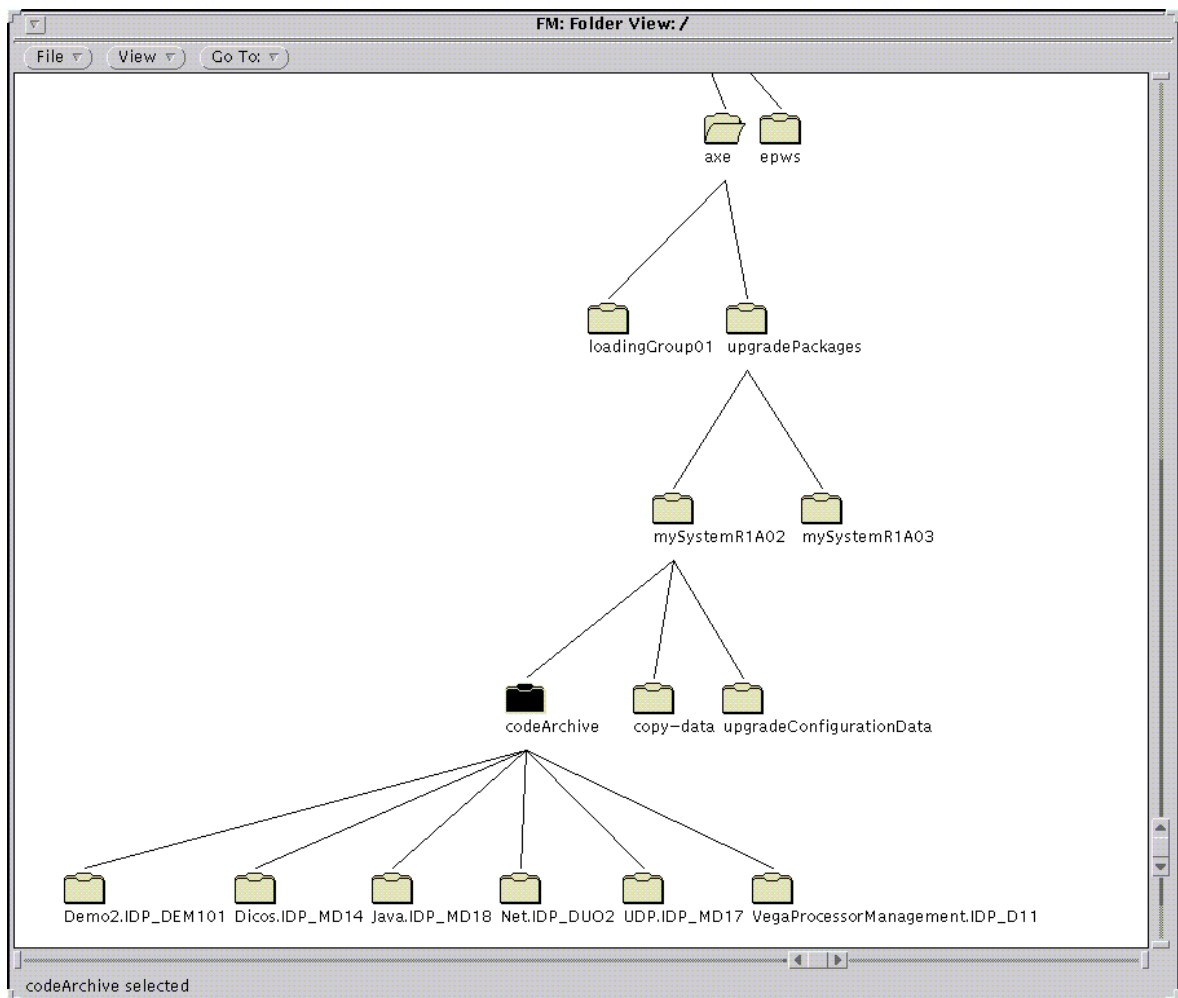


Figure 65. The file structure after building your upgrade package.

Those in Groups 1 and 2 will generate slightly different upGradePackages. The upGradePackages have to be tarred and then transferred to the IOs. It's also advisable to transfer the newly generated 'Site_for_DemoApp_Gx.config' files (from ~/<directory_name>/axe where the upGrade was built using the epct command) to the /opt/telorb/axe directory on both IOs.

Tar the Upgrade Package

```
%> cd ~/<directory_name>/axe/upGradePackages
%> tar -cvf UpGradePackage_G1 Site_for_DemoApp_G1
%> tar -cvf UpGradePackage_G2 Site_for_DemoApp_G2
```

Each person has generated their own upGradePackage and .tar file but only 2 of these will be used. The rest of the exercise is done in two groups. Each group handles their relevant upGradePackage.

The site configuration presently running on the cluster is one individual's that was loaded by both Groups in a previous exercise. It will therefore be necessary for one of the groups to generate its upGradePackage against that individual's 'site.config' file.

This involves copying a relevant 'Site_for_DemoApp_Gx.epct' into that individual's SiteConfiguration directory and running the upgrade command.

Take a BackUp

Group1 use the TelORB Manger to take a backup, name it 'Backup_Original.'

Rescan in the TelORB Manager to view the backup.

Each group can proceed with the following commands to place the upGradePackages on the IO machines.

```
%> ssh -l telorb <IP_Address_IO-x>
%> cd /opt/telorb/axe/upGradePackages
%> ftp <ip_address_local_WS>
%> pwd
%> cd ~/<directory_name>/axe/upGradePackages
%> bin
%> get UpGradePackage_G1 (UpGradePackage_G2)
%> get Site_for_DemoApp_Gx.config
%> bye
```

This places the 'UpgradePackage_Gx' tar file under '/opt/tel-orb/axe/upGradePackages' on IO1 and IO2. It places the 'Site_for_DemoApp_Gx.config' files there also.

Untar and create the loadingGroup directories

```
%> tar -xvf UpGradePackage_G1 (UpGradePackage_G2)
```

Wait for the untarring to finish and then remove the .tar file

```
%> rm UpGradePackage_G1 (UpGradePackage_G2)
```

Apply the Upgrades

This section of the exercises can only be done by **one group at a time**. Once Group1 have finished this section, Group2 will apply their UpgradePackage.

Before applying the Upgrade use the TelORB Manager to answer the following questions.

Question 57. How many backups are there?

Question 58. What pools are on Processors 1 and 7?

Processor1:_____

Processor7:_____

Now the software upgrade generated by Group_X can be applied.

- Rescan for Upgrades and select the upgrade package called "mySystem_for_DemoAppG_x" and click on the "apply" button under the "Upgrades" card.

- Wait for notification that the upGrade was applied successfully.
- Now the necessary software for DemoApp2000 is running on the system.
- Rescan in the TelORB Manager for Backups.

Question 59. How many backups are there? What are their names?

- Take a new backup, name it ‘Backup_Gx’. Observe the notifications of success or failure.

Question 60. What pools are on Processors 1 and 7?

Processor1:_____

Processor7:_____

Load UABDemoApp2000

- In the Loadable Extras on the ‘Home’ tab of the TelORB Manager select “UAB DemoApp 2000”
- Populate the Zone with 1000 subscribers, make sure you trace “pop/depop”.
- Set the print interval to 2 seconds.
- Set the Intensity of the traffic generator to 40 calls/sec
- Start the C++ traffic by clicking the RUN button, from now on every time we ask you to start or stop the traffic we mean the C++ traffic, unless differently specified.
- Press the Check Statistics button.

Question 61. Which processes are generating the printouts?
Why are we getting printouts in these windows?

Explain in terms of pools and process types.

- Make “Backup_Original” the active one.
- Reload Processor 7

Question 62. What pools are on Processor 7?

- Perform a Zone Reload

Question 63. What pools are now on Processor 7? Why?

Load UABDemoApp2000

- In the Loadable Extras select “UAB DemoApp 2000” and click ‘Show’.

Question 64. What happens? Why?
