UNIVERSIDAD DE LA REPÚBLICA
FACULTAD DE INGENIERÍA
INSTITUTO DE COMPUTACIÓN
PROYECTO DE GRADO
2003-2004

Gestión de Redes basada en Componentes

Integrantes:

Martín Giachino
Cristina González
Jorge Visca

Tutores:

Eduardo Grampín
Carlos Martínez

Agradecemos a: Federíco Rodríguez y Alexandra Castro.

ACERCA DE ESTE DOCUMENTO

El presente trabajo constituye la documentación correspondiente al Proyecto de Final de Carrera "Gestión de Red basada en componentes". El mismo fue realizado por los estudiantes Martín Giachino, Cristina González y Jorge Visca bajo la tutela del docente Eduardo Grampín durante el período Junio 2003 – Abril 2004. Se llevó adelante en paralelo con el proyecto "Gestión de Inventario". En conjunto, ambos proyectos son el punto de partida de un macro proyecto que tiene como objetivo principal implementar un sistema completo de Gestión de Redes.

La primera parte de este documento, está abocada a la presentación de los objetivos del proyecto. A continuación se realiza un análisis del estado del arte de las tecnologías, estándares y trabajos en el área de gestión de redes, y por último se presenta la documentación del sub sistema de Gestión de Red implementado (**mitiNum**). El contenido del documento se organiza en las siguientes secciones: Definición del proyecto, Analisis del problema, Plan de trabajo, Metodología y Proceso de Desarrollo, Estado del Arte, Analisis de la Arquitectura, Interaccion entre proyectos, Selección de Plataforma, Capa de Elemento, Capa de Red y Conclusiones. Además se complementa con una serie de apéndices.

ÍNDICE

ÍNDICE DE DIAGRAMAS

ÍNDICE DE TABLAS

1 DEFINICIÓN DEL PROYECTO

La gestión de las redes de telecomunicaciones se estructuran usualmente en un modelo de capas que comienza en el Nivel de Elemento Gestionado (dispositivos de la red), el Nivel de Red (visión global de la interconexión entre los Elementos gestionados), el Nivel de Servicio y el Nivel de Negocio. A esta estructura se la refiere usualmente como la "pirámide TMN".

Otra característica fundamental de los sistemas de gestión es la existencia de Áreas Funcionales: Gestión de Configuración, Fallos, Performance, Seguridad y Contabilidad (Accounting) (Fault, Configuration, Accounting, Performance, Security – FCAPS). Estas áreas funcionales son comunes a la estratificación TMN. Un sistema mínimo debe implementar al menos la gestión de Configuración, Fallos y Performance.

Esta estratificación y separación de áreas funcionales, junto al modelado de las redes con un enfoque orientado a objetos ha permitido gestionar eficientemente los servicios "tradicionales" de telecomunicaciones (telefonía). Por otro lado los servicios de datos basados en el protocolo IP ofrecido por los Proveedores de Servicio de Internet (ISPs) han sido gestionados tradicionalmente con un criterio menos formal, acorde al tipo de servicio "best-effort" que tradicionalmente han ofrecido estas redes. Sin embargo, la evolución de los servicios IP hacia un modelo de aplicaciones con garantías de Calidad de Servicio (QoS) a los usuarios impone la necesidad de desarrollar nuevas tecnologías de red y sistemas de gestión que permitan soportar esta evolución.

El presente proyecto estudiará las arquitecturas de gestión en conjunto con los paradigmas de desarrollo basado en componentes, e implementará algún(os) componente(s) básico(s).

1.1 Objetivos

Proponer una arquitectura de Gestión de Red basada en componentes e implementar el componente básico de Aprovisionamiento de Conectividad (Gestión de Configuración).

1.2 Resultados Esperados

El resultado fundamental es proponer una arquitectura de gestión basada en componentes, abierta a la integración hasta completar una solución integrada. A modo de "prueba de concepto" se debe implementar el módulo de Gestión de Configuración, la interfaz con el Nivel de Elemento y un prototipo de interfaz para la Gestión de Servicio.

1.3 Contexto de Trabajo

Se enmarca en las áreas de investigación del grupo de Arquitectura de Sistemas, específicamente en la disciplina de Gestión de Redes y Servicios.

2 ANALISIS DEL PROBLEMA

Un OSS (Operations Support System) es un sistema que realiza funciones de gestión, inventario, ingeniería, planeamiento y mantenimiento sobre redes de telecomunicaciones.

Originalmente, los OSS eran sistemas basados en mainframes diseñados para dar soporte a las compañías telefónicas, permitiéndoles realizar las operaciones de administración de las redes telefónicas de forma más eficiente. En la actualidad, los proveedores de servicio tienen que gestionar conjuntos de servicios y tecnologías de red mucho más complejas, por lo que los OSSs han evolucionado para afrontar los nuevos requerimientos. Las nuevas generaciones de OSSs usan lo último en tecnologías de la información para ofrecerles a las organizaciones formas más prácticas y útiles de gestionar sus servicios, ofreciendo altos niveles de calidad de servicios al cliente [1].

2.1 Requerimientos actuales

Una solución OSS tiene los siguientes componentes [1]:

- Motor de Workflow
  El motor de Workflow regula el flujo de información entre los distintos sistemas, controlando las distintas tareas que componen un proceso. Este motor puede ser propietario de la solución OSS, o un sistema de workflow general que se haya integrado.
- Ordering
  En este sistema es donde se ingresa toda la información necesaria para proveer servicios.
- Inventario
  El sistema de inventario mantiene toda la información acerca de instalaciones y equipos disponibles en su red.
- Diseño / ingeniería de circuitos y provisión
  Estos sistemas gestionan y rastrean los equipos y circuitos que físicamente proveen los servicios. Se encargan de especificar que equipos y rutas de red un servicio dado va a utilizar.
- Gestión y activación de elementos, y servicios de campo.
  Una vez que las tareas anteriores se completan, los servicios deben ser activados en la red. La activación puede consistir de varios pasos, que pueden incluir desde configurar remotamente un dispositivo, hasta dar la orden de despachar a un equipo de técnicos para realizar labores de campo.
- Gestión de Red y de Fallos
  Los sistemas de Gestión de Red son los encargados de la supervisión general de la red. Monitorean el tráfico de datos a través de la red y recolectan información estadística. También detectan e identifican los problemas y fallos. Una vez que un fallo es detectado, el sistema de Gestión de Fallos se encarga de encontrar e implementar una solución al problema, por ejemplo re-ruteando el tráfico.

Un OSS moderno debe convivir con multitud de servicios (enlaces telefónicos, ADSL, ISDN, T1…), dispositivos (redes ópticas, multiplexores, routers…) y lenguajes de control (CMIP, SNMP, TL1…), ofreciendo a la vez una gestión centralizada e integrada. La infraestructura computacional suele ser no-homogénea y distribuida, con productos de distintos fabricantes y diferentes arquitecturas, instaladas en ubicaciones físicamente separadas.

A su vez, nuevas tecnologías y estándares surgen continuamente, y deben poder ser integradas en la infraestructura existente. También existe la necesidad de mecanismos de interconexión entre organismos, de forma de implementar servicios que trasciendan el alcance de un proveedor de servicios.

3 Plan de trabajo, Metodología y Proceso de Desarrollo

El proyecto de Gestión de Red basada en componentes es de sus inicios un proyecto pensado para dos equipos (grupos) trabajando en paralelo por lo tanto una parte fundamental del mismo fue decidir como dividir el trabajo y como interactuar.

Una visión macro del plan de trabajo es la siguiente:

- Estudio del estado del arte: estudio realizado por ambos equipos en forma independiente, que incluye estudio de estándares.
- Selección de plataforma: investigación realizada por ambos equipos en forma independiente y acuerdo final según ventajas y desventajas. El acuerdo incluyó el lenguaje de programación y el ORB seleccionado[1].
- División del trabajo e interacción entre proyectos: identificación de componentes a implementar y asignación de los mismos a los equipos[2].
- Análisis, diseño, implementación de capa de elemento: implementación de la interface MTNM según la división de trabajo[3].
- Integración de capa de elemento: integración de los componentes implementados por cada equipo de trabajo.
- Análisis, diseño e implementación de capa de red: desición de la interface a utilizar teniendo en cuenta los objetivos asignados a cada equipo para capa de red. El equipo mitiNum debe implementar un componente Provisioning y el equipo Gesinv debe implementar un componente Inventario.
- Integración de todo el sistema: integración de los componentes de capa de red y de la capa de elemento.

Por otro lado la metodología y el proceso de desarrollo también surge desde los inicios debido a que una de las premisas era que fuera un desarrollo orientado a componentes.

A continuación describimos las etapas o fases del proyecto y para finalizar presentamos la bitácora del mismo.

Etapa cero

La *etapa cero* fue decisiva para el proyecto. En ella se realizó el estudio del estado del arte, se comenzó con el estudio de tecnologías, lenguaje de programación a emplear y estudio de los estándares. Durante esta etapa se discutió la división de las tareas de ambos grupos.

Primera etapa

Se puede decir que el proceso de desarrollo fue en fases iterativo, comenzando la primera etapa con una primera versión enfocados en el trabajo con la capa de elemento. Se dividió la capa en componentes y se asignaron componentes a los grupos.

Al finalizar esta *primera etapa* cada grupo tenía una primera versión de sus componentes y una aplicación gráfica cliente que permitió la prueba de los componentes de forma independiente. En esta etapa los componentes de ambos grupos no interactuaban entre si. El foco principal era lograr implementar las funcionalidades especificadas en la interface IDL (MTNM) para cada componente.

El hito que marco el final de la etapa fue una reunión donde cada grupo presentó lo que tenía implementado hasta el momento y los problemas de interacción que podrían surgir según sus respectivos puntos de vista.

---

1 Ver sección 7. Selección de Plataforma.

2 Ver sección 6. Interacción entre proyectos.

3 Ver sección 8. Capa de elemento.

Segunda etapa

Como *segunda etapa* se planteó como objetivo tener una primera versión de capa de elemento integrada. Es en este momento que surge la necesidad de agregar a la batería de interfaces existentes (IDLs de MTNM) una interface interna que permitiera el manejo de Cross Connet entre los componentes MLSNMgr de mitiNum y ManagedElementMgr de Gesinv[4]. Para poder introducir una interface útil, cada grupo puso sobre la mesa que necesitaba del otro. El objeto clave eran los Cross Connet, lo que no quedaba claro era que componente era responsable. Luego de analizar el problema se llegó a la conclusión que el componente responsable debía ser el ManagedElementMgr y que tenia que permitir al MLSNMgr setear Cross Connet para configurar una ruta. Como resultado de las discusiones el grupo Gesinv dueño del componente ManagedElementMgr liberó una IDL para uso interno de los mismos.

El resultado de la segunda etapa fue una versión integrada de capa de elemento donde era posible crear una SNC pasando por el componente MLSNMgr quien a su vez usaba las funcionalidades del componente EMSMgr de mitiNum y del componente ManagedElementMgr (las funcionalidades de MTNM y las de la nueva IDL).

Para integrar el prototipo de la segunda etapa no fue necesario una reunión intergrupal. Ambos grupos intercambiaron sus últimas versiones hasta el momento. Cabe aclarar que cada grupo compiló por separado las IDLs MTNM y que en el caso de la IDL interna la misma también fue compilada por separado una vez que fue liberada por el grupo Gesinv.

En paralelo a la segunda etapa de desarrollo se comenzó con el análisis y diseño de capa de red. Este punto fue más complicado ya que no teníamos en claro una interface como MTNM en capa de elemento. Sin embargo teníamos claro que el espíritu de interactuar entre grupos mediante interfaces estándares valía la pena ya que realmente nos había dado buenos resultados. Pudimos trabajar relativamente independientes y repartiéndonos a su vez el trabajo entre cada uno de los integrantes de los grupos.

Tercera etapa

La *tercera etapa* tubo como foco lograr una idea de que debería tener capa de red. Como parte de los objetivos del proyecto de grado el grupo mitiNum debía tener un componente Provisioning y el grupo Gesinv un componente Inventario.

Claramente ambos componentes debían interactuar para alcanzar sus objetivos. Por lo tanto nuevamente el foco fue ver que necesitaba cada componente del otro. En este caso el componente Provisioning seria el cliente del componente Inventario.

El Inventario por otro lado tenia que guardar por lo menos los objetos de capa de elemento y alguno más que surgiera en capa de red. En este mapeo fue muy útil el estudio del proyecto WINMAN[5] en el que se definen interfaces estándares para un inventario de red. También en dicho proyecto se utiliza CaSMIM para las funcionalidades de aprovisionamiento.

Provisioning sería además la cara visible para capa de servicio y por lo tanto se trató de definir una interface estandar hacia dicha capa. Se tomó como punto de partida CaSMIM (interface también del TMF pensada para dicho fin). De las funcionalidades cubiertas por esta interface surgen los nuevos objetos exclusivos de capa de red.

Ambos grupos trabajaron de forma independiente en las interfaces que iban a ofrecer, Provisioning en una versión adaptada de CaSMIM hacia capa de servicio e Inventario en una versión adaptada de WINMAN hacia Provisioning.

Cuando ambos grupos contaron con una versión que principalmente era una primera definición de las funcionalidades que ofrecerían, se realizó una reunión intergrupal de integración. En la reunión se trabajo en subgrupos formados por integrantes de cada grupo.

4 Ver sección 8.6 MultiLayerSubnetwork Manager

5 Ver sección 4.1.7 WINMAN

Un subgrupo verificó que las funcionalidades que esperaba Provisioning para poder implementar su propia interface hacia capa de servicio estuvieran presentes en el Inventario. Luego de esta reunión quedaron acordadas en un 90% las operaciones que ofrecería Inventario a Provisioning.

El otro subgrupo discutió el tema de la sincronización de los componentes de capa de elemento, sobre todo el tema de la inicialización de los mismos. Se decidió que cada componente mantendría la persistencia de sus objetos y que a su vez los componentes EMSMgr y ManagedElementMgr ofrecieran interfaces internas que permitieran crear, modificar y borrar sus objetos de forma de mantener la consistencia entre ellos[6].

La tercera etapa terminó con un acuerdo acerca de las funcionalidades del Inventario a utilizar por Provisioning y la desición de la especificación de dos nuevas interfaces internas.

Cuarta etapa

En la *cuarta etapa* cada grupo trato de cumplir el acuerdo de la tercer etapa.

El grupo Gesinv se abocó a liberar una primera versión que implementara cada una de los métodos y funciones de Inventario acordados.

El grupo mitiNum por su lado se abocó a implementar su interface a capa de servicio utilizando en cada punto que requiriera interacción con Inventario seudocódigo. Luego se paso a sustituir el seudocódigo por llamadas a las operaciones de un Inventario “trucho” donde las funciones no hacían mas que devolver mensajes. Gracias a esto se pudo avanzar con la lógica de Provisioning mientras se liberaba la primera versión funcional de Inventario.

La primera versión de Inventario fue liberada sin la necesidad de una nueva reunión intergrupal. Se sustituyó el Inventario “trucho” y se probó por primera vez la interacción entre Provisioning e Inventario.

Quinta etapa

Pasamos a la *quinta etapa* donde el objetivo fue liberar una versión integrada y completa de todo el sistema. Se definió como completa una versión que permitiera recibir un requerimiento en Provisioning de crear una connection, que este a su vez interactuara con el Inventario para obtener la información necesaria y que luego interactuara con capa de elemento para solicitar crear las SNCs según la ruta obtenida de un componente Routing[7] (mitiNum especificó una breve interface para tales efectos).

Para realizar ajustes se realizó una nueva reunión intergrupal donde se trabajó de forma similar a la reunión realizada en la tercer etapa.

Como resultado de la quinta etapa se obtuvo el sistema de Gestión de Red basada en componentes que le da nombre a nuestro proyecto de grado. El sistema está formado por la integración de los proyectos Gesinv y mtiNum. Desde esa fecha se ha ido mejorando, incorporando nuevas funcionalidades por parte de ambos grupos.

Es importante aclarar que además de las reuniones mencionadas se aprovecharon las reuniones de monitoreo para discutir los temas que surgían y que durante todo el proyecto nos mantuvimos en contacto por mail.

También hay que destacar que cada grupo contribuyó a la mejora de su par realizando testeos al utilizar los componentes del otro grupo.

6 Ver sección 8.5.4 Interface Interna.

7 Ver sección 9.8 Routing

Conclusiones

Podemos concluir que la forma de trabajo empleada: interfaces estándares, componentes, empleo de patrones de diseño introducidos tantos por las IDLs MTNM (iteradores, factory, etc) como por el lenguaje de programación utilizado (Java cuenta con un gran número de clases que se basan en patrones de diseño) permite trabajar en grupos de forma independiente. Incluso con grupos que se encontraran físicamente en lugares distantes ya que la cantidad de reuniones intergrupales fueron pocas.

Por otro lado fue muy importante para dividir el trabajo que los grupos fueran pequeños. Esto permitió en el caso de mitiNum, la realización de reuniones periódicas para dividir el trabajo y discutir soluciones a los problemas que se iban planteando. Además permitió compartir los resultados y experiencias de cada trabajo individual. De esta forma se evitó el re-trabajo y mucho código e ideas fueron reaprovechadas por todos los integrantes del grupo.

Bitácora

A continuación se presenta la bitácora del proyecto, donde solo se incluyen las reuniones intergrupales. Además de estas reuniones como se mencionó anteriormente nuestro grupo realizó reuniones periódicas con la participación de los tres integrantes.

Reunión	***Comentarios***
3/4/2003	Reunión de lanzamiento.
4/6/2003	Inicio formal del proyecto.
10/6/2003	Reunión de grupo. Presentes: Tutores e integrantes de mitNum.
1/7/2003	Primera reunión intergrupal. Se comienza a discutir la división e interacción entre ambos proyectos. Presentes: Tutores e integrantes de Gesinv y mitiNum.
16/7/2003	Reunión intergrupal para continuar con la discusión de división e interacción. Presentes: tutor Eduardo Grampín e integrantes de Gesinv y mitiNum.
29/7/2003	Reunión intergrupal (en el cuarto piso de facultad) para acordar la división definitiva a proponer a los tutores (sin tutores).
31/7/2003	Reunión intergrupal con Eduardo y presentación de la propuesta de división. En esta reunión no se define la división sino que queda a criterio de ambos grupos la decisión entre dos opciones (corte horizontal y corte vertical) y posterior confirmación al tutor.
15/8/2003	Reunión de avance con presencia de los tutores y ambos grupos.
21/8/2003	Reunión entre ambos grupos para acordar la división. Se acuerda corte vertical[8].
22/8/2003	Eduardo confirma formalmente por mail la división elegida (corte vertical).
22/9/2003	Reunión de monitoreo. Objetivo comentar el avance del proyecto a los tutores.
31/10/2003	Reunión de avance pedida por la comisión de proyectos del INCO. Demo de la primera versión del prototipo (solo capa de elemento sin integración entre grupos).
18/11/2003	Reunión de integración de capa de elemento. Se confirmo el uso de la interface interna para manejo de Cross Connect.
11/12/2003	Reunión para presentar la segunda versión del prototipo (capa de elemento integrada).
30/12/2003	Reunión de avance con ambos grupos y Eduardo.

8 Ver sección 6. Interacción entre proyectos.

Reunión	*Comentarios*
18/01/2004	Reunión de integración. Provisioning vs Inventario. Surge la necesidad de interfaces internas para inicialización de los componentes de capa de red.
18/02/2004	Reunión de ambos grupos con Eduardo para acordar fechas de entrega. Se acordó una presentación del prototipo para el 12/03/2004 y una entrega del borrador de la documentación para el 30/03/2004. Entrega final para el 12/04/2004.
04/03/2004	La comisión de proyectos del INCO aprueba la prorroga de un mes. Entrega final 12/04/2004.
06/03/2004	Reunión intergrupal para realizar ajustes al sistema.
16/03/2004	Reunión acordada para el 12/03/2004, demo del prototipo versión final. Presentes: Eduardo e integrantes de cada grupo.
30/03/2004	Se envió a Eduardo el borrador de la documentación por mail.
12/04/2004	Entrega final.

4 ESTADO DEL ARTE

4.1 Tecnologías relacionadas a la Gestión de Redes

4.1.1 TMN

En 1996, la CCITT (en la actualidad ITU-T) publicó un conjunto de recomendaciones para una arquitectura de sistemas de gestión de red, en la serie de recomendaciones M.3000 [5]. Cuando las redes de telecomunicaciones implementan las definiciones TMN [2], se vuelven interoperables, incluso cuando interactúan con dispositivos y redes de otros proveedores de servicios.

La arquitectura TMN y las interfaces fueron construidas sobre los estándares OSI. Estos estándares incluyen CMIP (Common Management Information Protocol), GDMO (Guideline for Definition of Managed Objects), ASN.1 (Abstract Syntax Notation One) y OSI reference model (modelo de referencia OSI de siete capas).

TMN es un framework orientado a objetos, que modela los recursos de red como objetos gestionados y la información gestionada de red como atributos de los mismos. Las funciones de gestión son desempeñadas por operaciones agrupadas en las primitivas CMIS (Common Management Information Service).

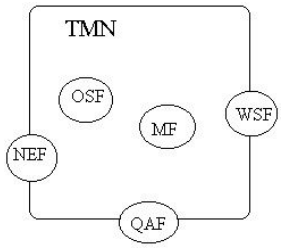
La información gestionada de red y las reglas que definen la forma en que esta información es presentada y gestionada es conocida como MIB (Management Information Base).

Los procesos que gestionan la información son llamados entidades gestionantes. Una entidad gestionante puede tomar dos roles posibles: gestor o agente. Los procesos gestores y agentes envían y reciben requerimientos y notificaciones utilizando el protocolo CMIP.

La arquitectura TMN tiene cuatro aspectos básicos o vistas que se pueden considerar separadamente, presentados en la recomendación M.3010:

- Arquitectura funcional

La arquitectura funcional describe la funcionalidad del TMN (ver Diagrama 1). Sus elementos son bloques funcionales y puntos de referencia. Los bloques funcionales son entidades conceptuales que permiten a TMN desempeñar las funciones de gestión (monitoreo, control, mediación, etc.). Los puntos de referencia representan el intercambio de información entre pares de bloques.



- Network Element Function (NEF) block
- Workstation Function (WSF) block
- Mediation Function (MF) block
- Q Adaptor Function (QAF) block

Diagrama 1 TMN: Arquitectura funcional

- Arquitectura de la información

La arquitectura de la información describe un encare orientado a objetos para el intercambio de información de forma transaccional en TMN. Esta reúne dos aspectos: un modelo de información gestionada y un intercambio de información gestionada. Ambos aspectos son adoptados de los estándares OSI.

  - El modelo de información gestionada representa una abstracción de los aspectos gestionados de los recursos de red y de las actividades de soporte relacionadas. El modelo determina el alcance de la información que puede ser intercambiada de una manera estandarizada.
  - El intercambio de información gestionada reúne las funciones: DCFs (Data Communications Functions) y MCFs (Message Communications Functions) que son utilizadas por componentes físicos para conectarse a la red de telecomunicaciones en una interface particular. Un MCF puede proveer las 7 capas de OSI, y puede tener interfaces para otras configuraciones de capas.

- Arquitectura física

La arquitectura física describe las interfaces realizables (ver Diagrama 2). Sus elementos son bloques constructivos e interfaces. Los bloques constructivos son distintos tipos de nodos físicos en TMN. Las interfaces definen el intercambio de información entre ellos. Los bloques constructivos generalmente son un mapeo uno a uno de los bloques funcionales aunque, un bloque constructivo puede contener mas de un bloque funcional del mismo tipo. Las interfaces son la implementación de los puntos de referencia de la arquitectura funcional. Una interface puede implementar uno o varios puntos de referencia del mismo tipo.

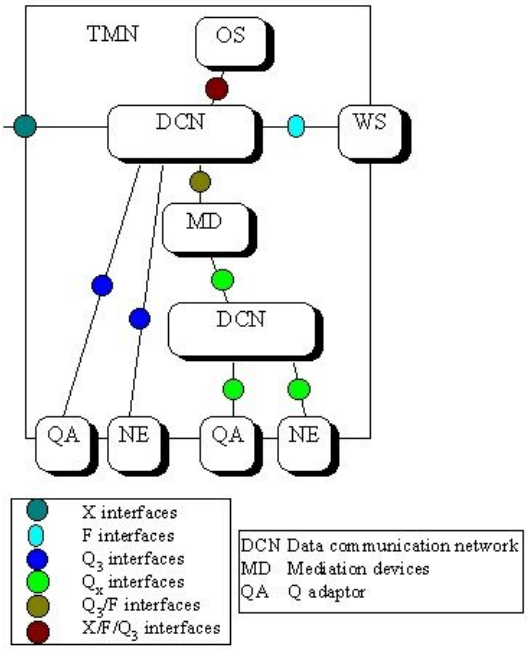


Diagrama 2 TMN: Arquitectura física

- Arquitectura lógica

La arquitectura lógica es un modelo jerárquico de cuatro capas que define los niveles de gestión de funcionalidades específicas (ver Diagrama 3). Los mismos tipos de funciones pueden ser implementados en muchos niveles, desde un alto nivel de negocios a un bajo nivel que maneja los recursos de red. Desde el nivel más bajo de la jerarquía se ubican las capas de gestión de elementos (EML), red (NML), servicios (SML) y negocios (BML), formando la llamada pirámide TMN. En la base de la pirámide se ubican los elementos de red (NE) que representan la información gestionada de TMN y los adaptadores (Q-adapter) que traducen la información TMN a información no TMN y viceversa.

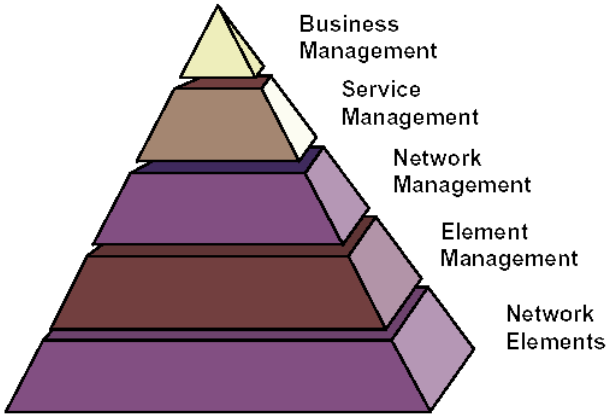


Diagrama 3 TMN: Arquitectura lógica

Esta visión de la arquitectura TMN esta muy vinculada a los protocolos OSI, en particular CMIP. El modelo OSI, si bien ofrece un modelo de información orientado a objetos, no es orientado a objetos en cuanto a su arquitectura de componentes, ni en sus comunicaciones entre componentes. Con el avance de las tecnologías de la información, surgieron métodos que se adaptaban mejor a la estructura orientada a objetos de TMN. También aumentaron los requerimientos de interoperabilidad por parte de los operados, aplicaciones de gestión y sistemas.

En consecuencia, TMN evolucionó en las siguientes direcciones [4]:

- Separación de la arquitectura del protocolo
- Mejor soporte para acceso externo
- Profundizar en la Gestión de Servicios
- Responder a los retos planteados por la evolución de Internet
- Explotar los desarrollos de OMG [10]
- Considerar la interoperabilidad entre componentes en el contexto de la industria de IT
- Adaptarse a tecnologías de red cambiantes.

Una primer etapa fue el incorporar CORBA sin abandonar CMIS/P como modelo de información. Soluciones de este tipo son BMP Server (CORBA-TMN bridge) y CORBA-TMN Gateway (JIDM Gateway). Estas soluciones permiten incorporar soluciones CMIS/P existentes, mediante encapsulamiento en objetos CORBA.

En una segunda etapa, se procedió a abandonar CMIS/P como modelo de información, y diseñar uno específico para CORBA. Entre las propuestas más interesantes en este sentido se encuentran la serie de recomendaciones de la ITU-T X.780 (Guidelines for Definition of CORBA Managed Objects), Q.816 (CORBA Based TMN Services), M.3120 (CORBA Generic Network and NE Level Information Model) y la propuesta del TMF 513 y 814 (MTNM Business Agreement). Estas propuestas coinciden en una solución independiente de los protocolos, distribuida, utilizando CORBA[5][6].

4.1.2 CMIS/P

El Common Management Information Services/Protocol (CMIS/P) ha sido estandarizado por ISO y recomendado por la ITU-T. CMIP implica el uso del stack de referencia OSI completo. CMIS/P es parte de OSI Managment, un conjunto de estándares que incluye GDMO, y otros protocolos. La recomendación ITU-T M.3010 es una aplicación directa de OSI Management para la gestión en el contexto de TMN. [6]

CMIS/P requiere altos niveles de especialización para modelado y diseño. Las herramientas que soportan esta tecnología no soportan una visión de alto nivel del dominio del problema. Por eso mismo, no se considera adecuado para tareas de gestión de servicios y de redes. Por otra parte, es una herramienta muy potente y difundida para la gestión de elementos [4].

El protocolo CMIS/P seguirá siendo usado debido a que existen muchas aplicaciones desarrolladas que lo utilizan, pero su posición es amenazada por nuevas tecnologías como CORBA y Java [9](JMAPI)

4.1.3 ITU-T M.3120 vs. TMF MTNM

La ITU-T a partir de su modelo de objetos basados en GDMO derivó un modelo basado en CORBA. En este modelo cada instancia de un objeto se representa en un objeto CORBA, definiendo una interface de granularidad fina.

Como los sistemas de gestión de redes manejan en general un gran volumen de objetos fue necesario incorporar una interface de granularidad gruesa. Para esto se agregó un conjunto de interfaces fachada donde cada una es un objeto CORBA y representa una clase.

El TMF propone un nuevo modelo de información a partir de la especificación del problema de negocio, especificación de requerimientos y casos de uso. Este modelo es de granularidad gruesa por fachadas donde cada una agrupa un conjunto de objetos del modelo de información de clases distintas pero con alto grado de cohesión.

Las ventajas sobre la propuesta de la ITU-T son:

- Ofrece operaciones y servicios que se enfocan en el problema principal a ser resuelto, de forma eficiente y atómica.
- El cliente (NMS) está protegido del modelo de objetos del EMS. Esto ofrece acoplamiento débil entre los objetos del NMS y el EMS.

4.1.4 CaSMIM

CaSMIM (Connection and Service Management Information Model) es una interface entre capa de servicio y capa de red propuesta por TeleManagement Forum [6].

El objetivo principal de CaSMIM es automatizar el aprovisionamiento y monitoreo de servicios end-to-end a través de redes multitecnologia, multivendedor y multicapa (multi-layered). Esto involucra traducción de requerimientos de servicios desde sistemas de gestión de servicios en acciones de aprovisionamiento de conexiones de red. Otra característica de CaSMIM es la de ser "orientado a servicio" pero "independiente de la tecnología" y por lo tanto ofrece flexibilidad para proveer conectividad a través de redes empleando tecnologías actuales y futuras.

4.1.5 SNMP

SNMP (Simple Network Management Protocol) [7] es un protocolo diseñado para facilitar el intercambio de información de administración entre dispositivos de red.

El sistema de SNMP consiste de tres partes :

1. SNMP manager (NMS)
2. SNMP agent
3. MIB

La idea para sacar provecho de SNMP, es que los NMS puedan recolectar datos automáticamente de los dispositivos de red. Para ello se utilizan dos métodos, traps y polling. Los traps loguean eventos críticos enviados por los SNMP agents al SNMP manager. Con el método de polling el SNMP manager hace polling a los SNMP agent para determinar el estado de los dispositivos de red. [*SNMP*]

Estos dos enfoques, pueden ser usados a la vez, no son disjuntos. Para decidir cual usar, hay que tener en cuenta diferencias importantes entre ellos. Una es el ancho de banda consumido: mientras el logueo de traps consume poco ancho de banda, el polling esta constantemente enviando y recibiendo datos.

Para el intercambio de datos (sea por polling o por traps), SNMP usa paquetes, en los cuales se llevan los valores de las variables definidas en las MIB's, y es necesario para ello, una clave única y compartida, la community strings.

SNMP es un servicio de capa de aplicación sin conexión, montado sobre TCP/IP, y que utiliza UDP. Imaginemos que si fuera orientado a conexión, y se usara polling, la cantidad de paquetes aumentaría mucho. Esto podría llevar también a que tal vez no sea el mejor para aplicaciones de tiempo real, ya que si un paquete que envía el NMS hacia el dispositivos de red se pierde, se tendría que retransmitir, lo que podría llevar a retardos no permitidos en este tipo de aplicaciones.

A continuación se presenta un diagrama con las operaciones que se pueden realizar con SNMP (ver Diagrama 4).

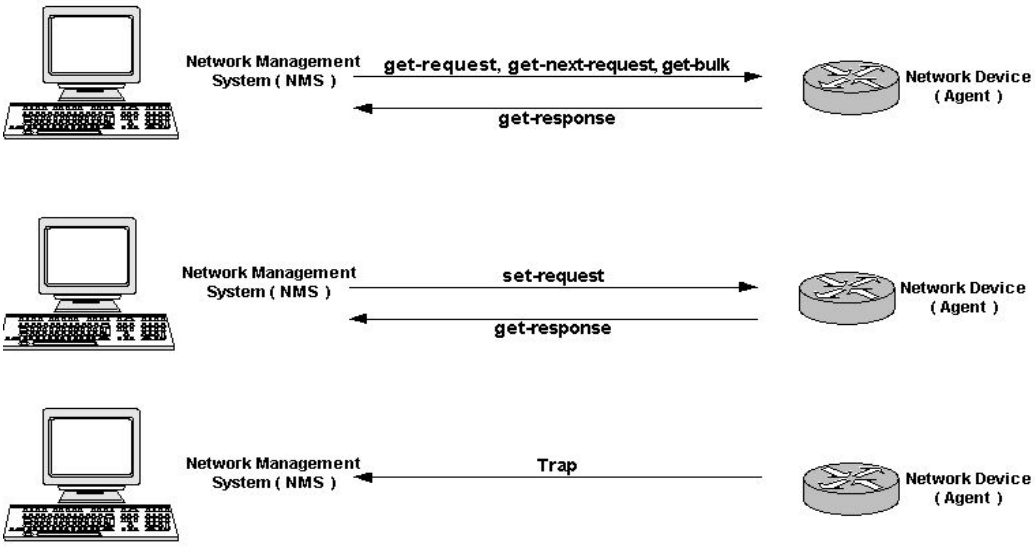


Diagrama 4 Esquema de operaciones en SNMP

Luego de ver las formas en que el sistema SNMP recolecta los datos, es hora de ver cuales son los datos.

Los datos se encuentran en el objeto administrado (dispositivo de red), en la MIB (Management Information Base) que esta en este, que es un árbol basado en una base de datos jerárquica de información.

Cada nodo del árbol tiene un string corto y un identificador entero. Siguiendo la secuencia de las etiquetas numéricas desde la raíz de árbol hasta el nodo, se obtiene el identificador del objeto (OID). El nombre del objeto es la secuencia de labels. Tener en cuenta que el OID es el identificador de una variable.

Esta estructura fue definida por organizaciones como la ISO, CCITT/ITU-T, y vendedores individuales. Además de lo mencionado, cada vendedor tiene un subárbol, con el conjunto de variables para sus equipos o servicios. Esto lleva a que las MIB's tengan una parte que es estándar y otra que no lo es.

En cuanto al acceso a estas variables, se definen tres tipos: read-only, read-write, write-only. Estos permisos son definidos en cada dispositivo de red, para cada community string que se acepte.

Este punto es uno de los puntos más débiles de este sistema, en lo que tiene que ver con la seguridad. Por eso este es uno de los aspectos modificados en SNMPv3.

4.1.6 TINA

La arquitectura TINA es fruto del trabajo del consorcio TINA-C (Telecommunications Information Networking Architecture Consortium) que existe desde los 90. Su fuerte es la gestión de servicios; su propuesta se considera mucho más amplia que la propuesta de TMN. [*8*]

En TINA las capas de gestión de TMN fueron generalizadas en tres capas: servicios, recursos y elementos. Las capas son usadas para definir el alcance de las actividades de gestión en la arquitectura de servicio y en la arquitectura de red. La arquitectura de servicio de TINA puede ser mapeada en la gestión de servicio de TMN, no así la arquitectura de red que cubre tanto la gestión de red como la gestión de elemento de TMN. Si bien TINA toma la mayor parte de los conceptos de TMN no condiciona el uso de CMIS/P, GDMO o el modelo de información de gestión de redes de la ITU-T. Por el contrario TINA es uno de los emprendimientos que adoptó la visión de que los desarrollos de sistemas de telecomunicaciones deberían usar encares ODP (Open Distributed Processing). Es éste uno de los mayores aportes de TINA al mundo TMN junto con la definición del concepto de DPE (Distributed processing environment) que luego se traslado a CORBA. [3]

4.1.7 WINMAN

El proyecto WINMAN [17] (WDM and IP Network MANagement) es un proyecto europeo (julio 2000 – marzo 2003) de investigación y desarrollo, que tiene como objetivo ofrecer una solución integrada de gestión de redes para el aprovisionamiento y mantenimiento de servicios de conectividad IP sobre redes WDM (Wavelength Division Multiplexing) de alta capacidad.

WINMAN se basa en dos tecnologías emergentes para la gestión de conexiones end-to-end:

- WDM: optimiza el uso del ancho de banda en fibra óptica.
- MPLS (Multi-Protocol Label Switching): brinda QoS e ingeniería de tráfico para el mundo IP.

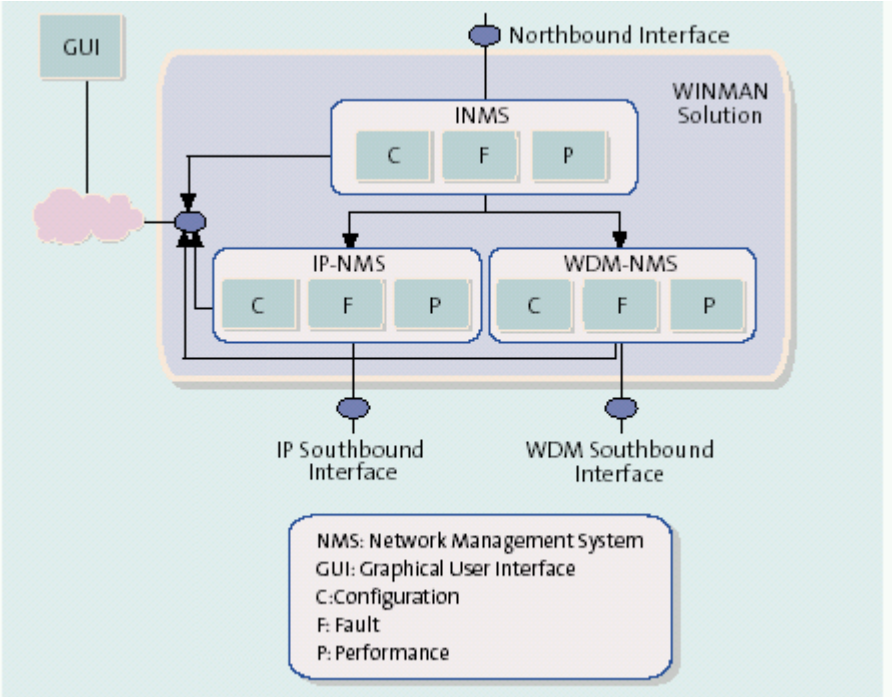


Diagrama 5 Arquitectura WINMAN

La solución WINMAN se ubica en la NML (Network Management Layer) de la pirámide TMN. La arquitectura WINMAN tiene dos puntos de referencia para la interacción con sistemas externos:

- Interface norte (Northbound interface): orientada hacia los sistemas de gestión de servicios (Service Management Systems), tales como VPN y sistemas de VoIP o MoIP.
- Interface sur (Southbound interface): orientada hacia los sistemas de gestión de elemento (Element Management Systems).

WINMAN utiliza dos estándares del TeleMangement Forum para las interfaces mencionadas:

- CaSMIM: como interface entre WINMAN y las aplicaciones de capa de servicio.
- MTNM: como interface entre WINMAN y los sistemas de gestión de capa de elemento (EMSs).

4.2 Tecnologías de la Información para Ambientes Distribuidos

4.2.1 Java

Java es un lenguaje de programación orientado a objetos, independiente de la plataforma, perfectamente dotado para su utilización en Internet.

Su integración con CORBA, RMI o COM+ brinda ambientes distribuidos de objetos, y con JMAPI (Java Management Application Programming Interface) permite desarrollos fáciles y rápidos de aplicaciones de gestión distribuidas. JMAPI facilita la creación de applets para gestión de redes e incluye un conjunto de utilidades para mapear SNMP a Java. [*9*]

Java junto con CORBA son una buena opción para soportar la necesidad de comunicaciones de una solución web TMN. Al utilizarlos, los usuarios tienen acceso a la información de forma transparente, sin tener que conocer sobre que plataforma de software o hardware residen o en que ubicación dentro de una red de comunicaciones. Una alternativa a CORBA puede ser una solución RMI (Remote Method Invocation), RMI es eficaz y fácil de utilizar, siempre que la comunicación sea solamente entre aplicaciones Java. Por otro lado otra alternativa posible puede ser la integración de Java con COM+, si estamos en un ambiente Microsoft. [4]

Otras ventajas de Java:

- Gracias a su gran popularidad en los últimos años se desarrollaron gran cantidad de APIs y herramientas de soporte.
- Ofrece un entorno de programación fácil de usar y entender.
- Disponibilidad de muchos desarrolladores experimentados.

4.2.2 CORBA

CORBA, Common Request Broker Architecture pertenece a un conjunto de tecnologías creadas por el grupo OMG (Object Management Group) con el propósito de diseñar un framework que permita la implementación de componentes de software distribuidos, portables, reusables e interoperables, basados en interfaces estándar orientadas a objetos [10].

La arquitectura CORBA está compuesta de un bus de software conocido como ORB (Object Request Broker) al que se conectan los clientes CORBA que referencian objetos remotos y los servidores CORBA que tienen los objetos solicitados por los clientes. Los clientes y los servidores comparten una única definición para sus interfaces, especificada en IDL (Interface Definition Language). IDL es independiente del lenguaje de implementación.

La arquitectura CORBA ofrece además un conjunto de servicios que aumentan la funcionalidad del ORB como son el Servicio de Nombres, el Servicio de Notificación entre otros.

En el mercado existen muchas implementaciones CORBA que se diferencian en los servicios que brindan, los lenguajes que soportan, los tipos de licencias, plataformas sobre las que corren, etc. Entre los más difundidos se encuentran: VisiBroker, TAO, JacORB [18], Orbix y MICO.

Motivaciones para usar CORBA [11]:

- CORBA/IDL por ser orientado a objetos se adapta bien al modelado
- SNMP es ineficiente debido a su simplicidad y falta de orientación a objetos
- CMIP/GDMO es demasiado complejo, caro e infrecuente
- Como CORBA es un estándar de IT de uso general ampliamente difundido:
    - Las herramientas son de menor costo y hay mayor oferta.
    - Hay mayor cantidad de técnicos capacitados, mayor experiencia y más literatura

Estándares de gestión CORBA [11]:

- JIDM (Joint Inter-Domain Management), mapeos desde SNMP y GDMO/CMIP realizados por TMF, OMG y X/Open.
- Estándares CORBA nativos, desarrollados por ITU-T, TMF, TINA-C, entre otros.

4.2.3 UML

UML (Unified Modelling Language) es un lenguaje para especificar, visualizar, construir y documentar cada una de las partes de los sistemas de software, así como también para construir modelos de negocio y modelar otros sistemas distintos de los de software. Esta herramienta ha sido usada por la mayoría de instituciones involucradas en el desarrollo de tecnologías de gestión de redes, y ha sido promovida por la OMG [10].

4.2.4 XML

XML (eXtensible Markup Language) es un metalenguaje: es un lenguaje para definir lenguajes. Los elementos que lo componen pueden dar información sobre lo que contienen, no necesariamente sobre su estructura física o presentación, como ocurre en HTML.

XML no nació sólo para su aplicación en Internet, sino que se propone como lenguaje de bajo nivel (a nivel de aplicación, no de programación) para intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo, etc.

Motivaciones para usar XML [16]:

- Es extensible, independiente de la plataforma, y soporta internacionalización y localización.
- Las reglas de XML son estrictas y permiten especificar por ejemplo, archivos de configuración sin ambigüedades.
- XML es una familia de tecnologías. Es un conjunto creciente de módulos que ofrecen servicios útiles para realizar tareas importantes frecuentemente demandadas (DTD, CSS, XSD, etc).
- Permite definir un formato de documento combinando y reusando otros formatos.
- Es gratuito, independiente de la plataforma y bien soportado.

4.4 Productos

4.4.1 HP Open View – Network Node Manager

HP Open View Network Node Manager (HPOV) provee básicamente un mapa de las redes, muestra el estado de sus objetos, y ofrece varias herramientas para administrarlos [12]. Entre las funciones más importantes se encuentran:

- Descubrimiento de la red
- Administrador de Eventos
- Actualización automática del mapa de estado
- Administrador de fallas

Veamos los componentes básicos:

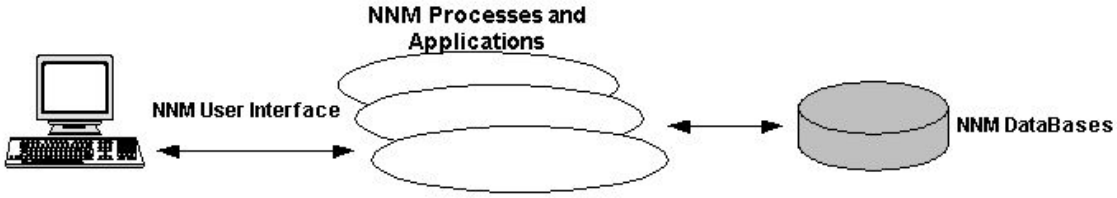


Diagrama 6 Componentes del HP Open View

Los “NNM Proceses and Applications” se dividen en dos, los que se activan cuando arranca el NNM y los que son activados por el usuario. Los primeros son los responsables de recolectar y guardar datos en todo momento, y hasta que se detenga el NNM.

En cuanto a las NNM DataBases se ven tres tipos, pero sólo dos de ellas son relevantes para nuestro estudio. Ellas son la Object Database y la Topology Database.

La primera de ellas guarda información genérica de cada objeto descubierto, como capacidades y atributos. La segunda guarda información crítica como el IP status e IP forwarding.

Hablando ahora sobre el funcionamiento del NNM, decimos que cuando se inicia comienza con el Descubrimiento de los nodos de la red. Este proceso por defecto solo descubre nodos que se encuentran en la misma red, marcando los routers como nodos no manejados. Luego estos nodos se pueden manejar por pedido del usuario, entonces se comienza con el descubrimiento de esa red.

Este proceso se hace vía SNMP, pidiendo a cada nodo información de su configuración como dirección IP, máscara de red, y Caché ARP. Esto pone de alguna manera límites sobre los dispositivos de red que se pueden descubrir, ya que deben ser nodos IP, y que se puedan administrar vía SNMP.

Otros procesos le permiten a NNM monitorear la red. Se puede decir que tiene componentes de Fault, Configuration y Performance, de los cuales el de Fault y Configuration son meramente de consulta. Si bien se puede saber el estado y la salud de la red, estos no disparan acciones de recovery, sino que solo le avisan al usuario de los problemas y es este quien emprenderá las tareas de corrección.

En cuanto a los procesos de monitoreo, NNM usa el método de polling para sus cuatro tipos de monitoreo, que son: Configuración General, IP Discovery Configuration, Status Polling Configuration y Secondary Failure Configuration.

NNM también permite cargar nuevas definiciones de MIB, según sean necesarias para manejar algunos dispositivos. Estas nuevas MIB deben estar definidas bajo el estándar ASN.1.

4.4.2 Cisco View 4.2

Es una aplicación para la administración de dispositivos de red. Para ello se basa en la interfaz MIB II, la cual le permite monitorear parámetros para WANs, como pueden ser Frame Relay y X.25, LANs, y otros protocolos de red. También provee de un inventario con datos sobre versiones de software, números seriales; configuraciones de hardware como memoria del sistema, flash, versiones de firmware, y MAC adresses [13].

Además puede ser usado para realizar varios niveles de configuración de switches y routers.

Para la recolección de los datos y el mantenimiento de ellos, Cisco View utiliza SNMP, haciendo polling sobre variables Cisco de la MIB.

Cisco View es un gestor que se puede comunicar con todos los Cisco Devices, como routers, LAN switches, ATM switches, etc.

Sobre las interfaces Frame Relay se muestra la lista de DLCIs activos en la interfaz, y se puede crear, borrar o modificar circuitos extras.

Por supuesto que Cisco View tiene una amplia variedad de variables monitoreadas, utilización de CPU, memoria disponible, buffers.

También brinda estadísticas sobre utilización, broadcast, errores, etc.

Sobre las interfaces con interfaces Frame Relay, se pueden monitorear los PVCs por separado.

Si bien esta aplicación es un poco más potente que la anterior, en el sentido que da una vista más amplia e integrada de la red, tiene como inconveniente que esta pensado para dispositivos Cisco propietarios.

4.4.3 Resource Manager Essentials (RME)

Este paquete provee una solución empresarial para la administración de redes, sobre dispositivos Cisco.

Entre los rasgos más importantes, y diferentes a los casos anteriores tenemos la configuración de dispositivos y la administración de software. Cosas que estarían dentro de las funciones que realiza el componente “Network Inventory Management Process” del TOM Process.

Un aspecto interesante es el de poder importar el inventario desde archivos, o desde otro NMS, ya sea local o remoto.

4.4.4 CWSI Campus 2.2

Campus es una solución potente que reúne las funcionalidades vistas por los productos anteriores y los extiende.

Entre lo destacable del Campus 2.2 tenemos la integración de dispositivos ATM, lo que trae aparejado que se tenga mas control sobre parámetros de QoS.

En el Diagrama 7 se presenta el Framework sobre el que está basado el producto.

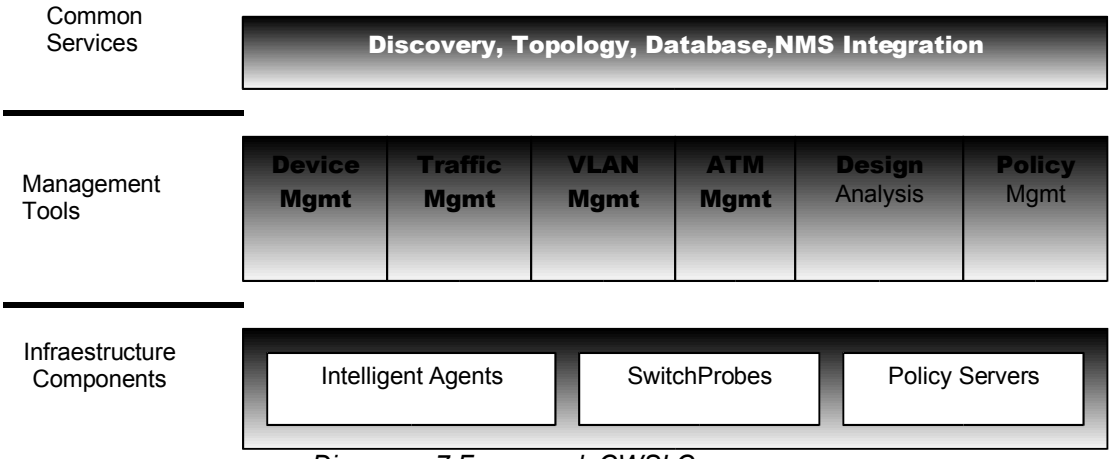


Diagrama 7 Framework CWSI Campus

En Campus se han aumentado las tecnologías en los agentes, entre los que encontramos el clásico SNMP, RMON, RMON2, usados para el monitoreo de trafico, ILMI para el descubrimiento de ATM, todos ellos estándares de la industria. Por otro lado, hay extensiones propias como CDP (Cisco Discovery Protocol), ISL, DISL, VTP y VQTP.

Aplicaciones que componen CWSI:

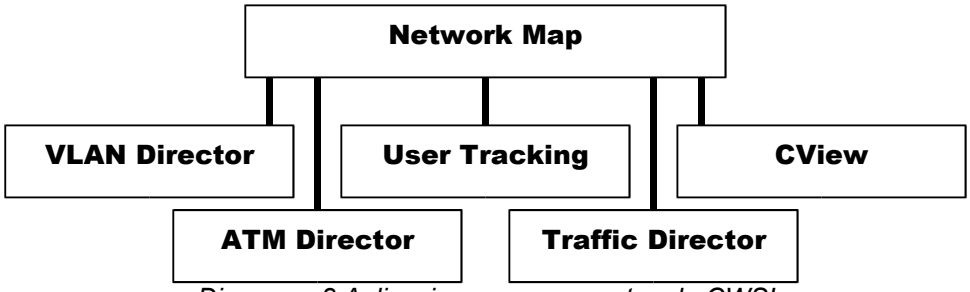


Diagrama 8 Aplicaciones componentes de CWSI

De las anteriores aplicaciones, veremos las dos más interesantes: ATM Director y VLAN Director, ya que ellas son las dos que tocan mas de cerca la administración de conexiones end-to-end y QoS.

Comenzando por el ATM Director, digamos que es el que se encarga de el descubrimiento de switches ATM, addresses y dispositivos neighbor. Por otro lado, también se encarga de el rastreo de PVCs, análisis de circuitos layered on top.

Continuando con VLAN Director, un componente muy poderoso, digamos que no solo trabaja con la red en su forma “física”, sino que también permite un manejo lógico de las VLANs. Permite su creación, configuración y monitoreo, e incluso avisando sobre discrepancias y errores de configuración. Es posible además configurar LANE services y crear ATM VLANs.

4.4.5 ALCATEL 5620 NM

Sin dudas, uno de los productos más potentes en cuanto a la integración de diferentes redes, y la inclusión de administración de conexiones end-to-end [14].

Este producto nace originalmente como una plataforma de gestión de servicios TDM, para luego extenderse a ATM, Frame Relay, ADSL y estaría llegando a MPLS, lo que le da un perfil más integrador.

ALCATEL 5620 es capaz de manejar distintas tecnologías de WANs, como IP/MPLS, ATM, Frame Relay, SONET/SDH, TDM y X.25 y X.75, y diferentes *access technologies*, como DSL y wireless.

4.4.6 AdventNet TMF EMS/NMS

AdventNet TMF EMS y NMS son dos productos que implementan la interface NML-EML espacificada por TMF513/608/814, versión 2.5. Proveen un framework que permite el desarrollo de aplicaciones, ofreciendo los servicios de gestión básicos especificados por TMF, como gestión de inventario, aprovisionamiento, gestión de fallos, etc [15].

AdventNet TMF EMS

Se dirige a dos perfiles de proveedores de software para EMS:

- Vendedores de equipamiento: AdventNet TMF EMS soporta el desarrollo de EMSs para NE, y de sus interfaces hacia el NML
- Vendedores Independientes de Software (ISV): AdventNet TMF EMS soporta el desarrollo de soluciones EMS llave en mano.

AdventNet TMF EMS está construido sobre la plataforma AdventNet TMF NMS.

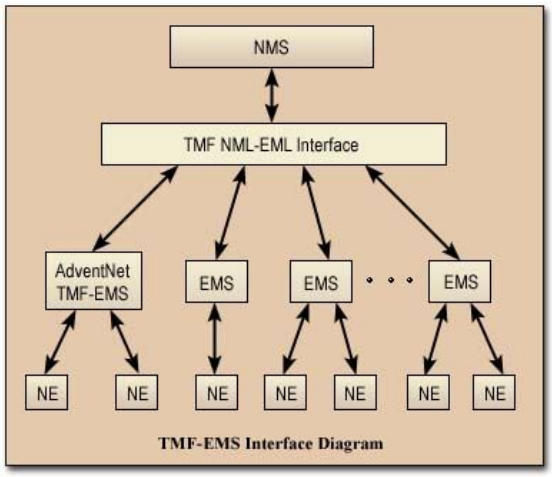


Diagrama 9 AdventNet TMF EMS

AdventNet TMF NMS

Se dirige a dos perfiles:

1 Vendedores Independientes de Software (ISV), Integradores y empresas: pueden construir aplicaciones “out-of-the-box”, systemas y soluciones de gestión de red para redes compatibles con TMF.

2 Proveedores de Servicios: pueden contruir OSSs.

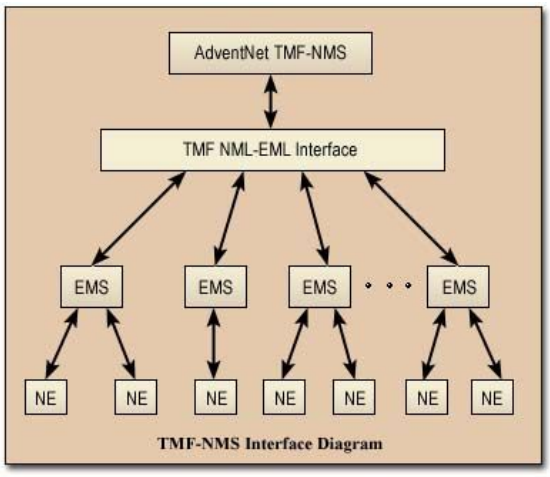


Diagrama 10 AdventNet TMF NMS

4.5 Conclusiones del Estado del Arte

Se propone enfocar el estudio en sistemas TMN distribuidos sobre plataforma CORBA, con componentes escritos en Java. Se usará MTNM versión 2 como la interface entre capa de red y capa de elemento.

Se analizará la especificación de MTNM, para decidir cómo se aplica el alcance acordado para el proyecto sobre el conjunto de componentes e interfaces.

Además se realizará un prototipo entre capa de servicio y capa de red tomando como base la especificación de requerimientos propuesta en CaSMIM, asi como también su conjunto de IDLs. Se tendrá en cuenta únicamente la arquitectura y las estructuras de datos que se aplican al alcance del proyecto.

Para no limitar las vías en las que el sistema pueda desarrollarse en el futuro, y minimizar el esfuerzo necesario para integrar nuevos componentes y funcionalidades, se usará únicamente comunicación CORBA, omitiendo otras tecnologías de componentes. Toda la información que tenga que transmitirse o almacenarse que no pase por CORBA, se manejará como archivos XML.

5 Analisis de la Arquitectura

En el marco de nuestro proyecto y cubriendo los objetivos del mismo se desarrollo **mitiNum**. Este es un sistema de gestión de redes que cubre el área funcional de aprovisionamiento y que está estratificado en capas. De cada capa se implementó la parte íntimamente relacionado con gestión de aprovisionamiento y se interactuó con el sistema Gesinv para servirnos de funcionalidades de inventario.

Veremos a continuación el sistema **mitiNum** desde diferentes vistas de la arquitectura.

5.1 Capas

En este apartado se intentará dar una descripción de la arquitectura en capas del sistema, sin ahondar en la implementación concreta de las interfaces que las conectan, o en las funcionalidades brindadas por las mismas, sino que pretende fijar los conceptos o estándares en los cuales están basadas. La intención concreta es mostrar las bondades de la arquitectura en capas y el desarrollo basado en estándares.

Como la mayoría de las aplicaciones comerciales de gestión en telecomunicaciones que existen en el mercado[9], el sistema se basa en la arquitectura de capas como especifica la pirámide TMN[10] (ver Diagrama 11).

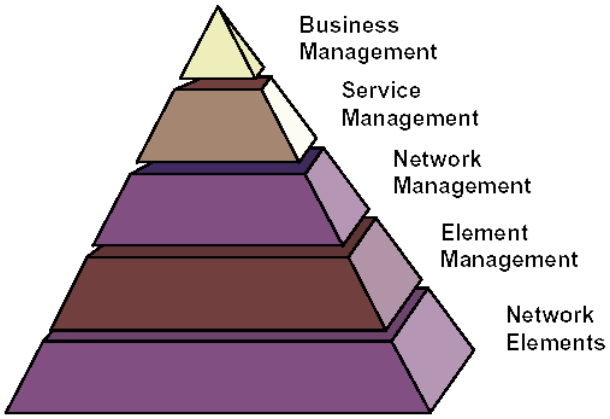


Diagrama 11 TMN: Capas

En la pirámide las capas están discriminadas según las funciones y objetivos que intentan cubrir, y que son de interés a las empresas que usan este tipo de sistemas. Esta división en capas permite tener diferentes vistas de la red que se corresponden con las necesidades de distintas áreas de las organizaciones y empresas, y que permite tener independencia entre ellas. De esta manera, los sistemas concebidos usando esta arquitectura permiten mantener aisladas las funcionalidades de cada capa, y la posibilidad de un fácil intercambio de las mismas, conforme hayan mejoras sustanciales en alguna de ellas.

A fin de ejemplificar lo comentado hasta aquí, describiremos un posible escenario que atañe a cada capa en una empresa de telecomunicaciones. Por un lado tenemos personas encargadas de trabajar con los dispositivos físicos (routers, switches, etc), que estarían contemplados en la capa de elemento (Element Management), y por otra parte tenemos personal encargado de la configuración de la red, sin que esto implique el contacto de estos con los dispositivos que están bajo la supervisión del personal anterior. Luego tendríamos operarios que crean servicios sobre la red (Service Management), servicios que podrían haber surgido de decisiones gerenciales, tomadas con la ayuda de la capa de Business Management a partir de datawarehousing. Claramente vemos entonces, el interés de algunos usuarios en alguna de las visiones según su función.

9 Ver ejemplos en el Estado del arte, sección 4.3 Productos.

10 Ver Pirámide TMN en el Estado del arte, sección 4.1.1 TMN.

Explicado el contexto en el que se encuentra, veamos ahora la arquitectura del sistema **mitiNum.** El sistema esta concebido de acuerdo a una arquitectura como la que muestra la pirámide TMN, de manera que hereda las bondades mencionadas anteriormente, pero además implementa cada capa según estándares abiertos como son MTNM y CaSMIM[11]. Esto permite que el cambio sea aún mas transparente, e incluya cualquier otro fabricante que respete el estándar.

El sistema **mitiNum** implementa en forma parcial algunas de la capas mencionadas, como se describe a continuación.

La capa inferior, Network Elements, es resuelta por el grupo Gesinv[12], que simula los dispositivos de red como pequeños programas de software, que básicamente permiten la creación de cross-connections sobre ellos, y el seteo de ciertos valores. Estos dispositivos necesitan de drivers que deben ser provistos, de forma que la capa de elemento pueda interactuar con ellos.

La capa de elemento (Element Management) se encarga principalmente de mantener una correspondencia en software de la red física subyacente, y de proveer de todos los datos necesarios a la capa superior (Network Management). Para mantener la correspondencia con la red física se vale de los drivers para leer la información de los dispositivos que la forman, y a partir de ellos servir los requerimientos que llegan desde la capa de red. Estos requerimientos de capa de red se hacen a través de la interface que la capa de elemento exporta, y que en **mitiNum** es la NML-EML Interface Version 2.0[13].

La capa de red (Network Management) en **mitiNum** se vale de dos estándares para la comunicación con sus capas vecinas. Uno de ellos ha sido mencionado anteriormente, NML-EML Interface Version 2.0, y es usado para interactuar con la capa de elemento. El restante permite el diálogo con la capa de servicio, de manera que esta última pueda realizar peticiones. Cabe acotar en este punto que lo que se usa no es exactamente el estándar, sino una adaptación del mismo, de acuerdo a lo acordado en el alcance del Proyecto. Este segundo estándar es CaSMIM, que está definido en el documento TMF807 versión 1.5[14] y la versión adaptada para nuestro proyecto fue denominada miCasmim.

Para terminar de comentar la arquitectura de **mitiNum** queda mencionar que el sistema implementa un cliente de capa de red, y no exactamente una capa de servicio. Esto le permite al prototipo trabajar con la capa de red. Este cliente trabaja simulando la capa de Servicio, enviando pedidos a la capa de red, usando miCasmim como interface, pero sin tener interface con la capa superior, Business Management.

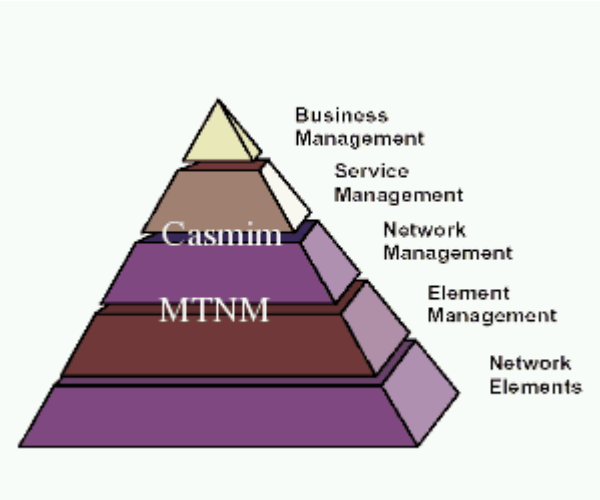


Diagrama 12 Interfaces del TMF

11 Por detalles de implementación de las interfaces ver secciones 8.Capa de Elemento y 9.Capa de Red.

12 Ver documentación del proyecto Gesinv.

13 Ver documento de MTNM TMF513V2_0.pdf.

14 Ver documento de CaSMIM TMF807v1_5.pdf.

5.2 TOM

Una vez decidida la arquitectura global del sistema, en este caso una arquitectura en capas como se describió en el apartado anterior, comenzamos ahora a internarnos en la arquitectura de cada capa en particular.

Recordamos la definición de TOM (Telecom Operations Map). Este es un framework que permite la automatización *end-to-end* de los procesos de las telecomunicaciones y servicios de datos[15]. Este framework plasma en su esquema los procesos que serían necesarios para proveer servicios *end-to-end*, discriminados según las capas de la pirámide TMN. Es importante mencionar que TOM intenta dar soporte a las áreas funcionales de las telecomunicaciones (FCAPS).

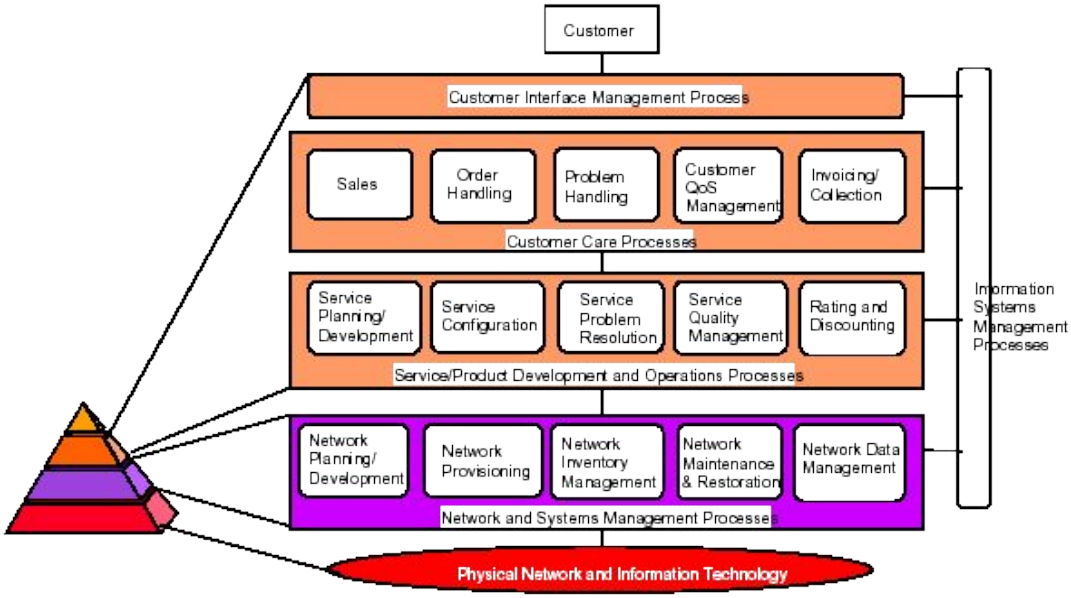


Diagrama 13 Telecom Operations Map

El sistema **mitiNum**, en conjunto con el proyecto Gesinv, abarca algunos de estos procesos que identifican áreas funcionales, que de alguna manera son heredados del TOM. **MitiNum** cuenta en su arquitectura con dos de estos procesos en cada capa que implementa, Element Management y Network Management (ver Diagrama 14).

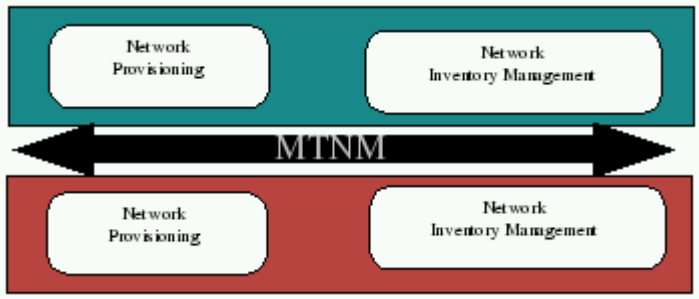


Diagrama 14 Procesos del TOM incluidos en el sistema

15 Por mas detalles ver documento de MTNM TMF513V2_0.pdf.

5.3 Interfaces

Continuando con la descripción de la arquitectura de **mitiNum**, se verá en este apartado las interfaces usadas para la comunicación entre distintas capas con más detalle. Estas interfaces tienen implicancias importantes a nivel de la arquitectura, ya que al estar definidas en idls CORBA, claramente nos lleva a un contexto de una arquitectura distribuida y basada en componentes. Un sistema construido sobre este sistema puede lograr altos niveles de interoperabilidad entre distintas plataformas y lenguajes[16].

MitiNum implementa MTNM de acuerdo a lo especificado en el documento TMF513V2_0.pdf. Esta versión es la Public Evaluation, por lo que puede estar sujeta a correcciones hasta su liberación final, y puede contener errores. Actualmente el estándar se encuentra en la versión 3.0, versión que fue liberada durante el transcurso del proyecto. Se decidió continuar con el desarrollo basado en la versión 2.0, dado el avance del proyecto. Esta interface esta especificada por un set de archivos idls CORBA[17]. Estos archivos se pueden ver en TMF814v2_0/idl_v2.0.

Centrando ahora la atención en la capa de elemento, y haciendo una descripción más detallada, en el siguiente diagrama se presentan las interfaces CORBA que **mitiNum** exporta en capa de elemento en la figura que sigue. Este conjunto de interfaces CORBA se divide en dos grupos, las root interfaces (parte superior) y las fachadas en la parte inferior. Esta distinción entre interfaces tiene que ver con la función que cada una lleva a cabo[18], aunque no es la única diferencia. Para una comprensión más profunda de este tema, se recomienda leer el documento TMFC2002 Using-CORBA-for-MTNM-12-May-2003.pdf, que explica muy bien ciertos puntos y en el cual se basan algunas de las decisiones tomadas.

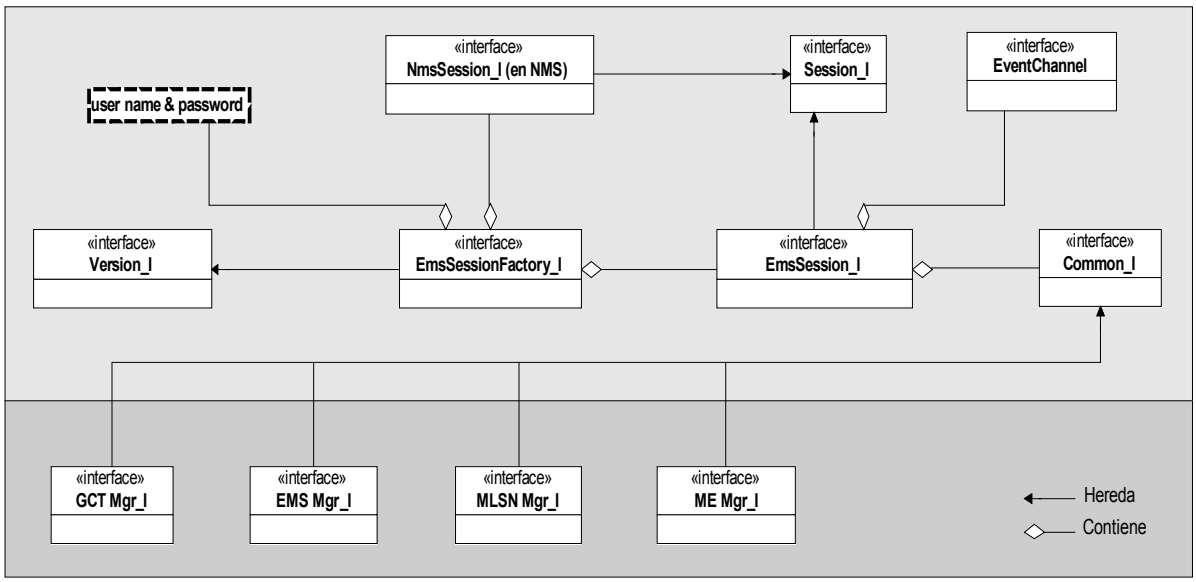


Diagrama 15 Root interface

Mirando el Diagrama 15 podemos inferir algo sobre el normal funcionamiento de la interacción entre capa de red y capa de elemento, en donde la capa de red se conecta en principio a las root interfaces a modo de abrir una sesión, para luego tener acceso a las fachadas y por medio de estas realizar las funciones sobre los elementos de la red.

16 Por mas información ver Estado del Arte sección 4.2.2 CORBA y http://www.omg.org

17 CORBA es un estándar abierto, definido por el OMG http://www.omg.org

18 Ver sección 8.2 Sesiones.

Hay que mencionar que **mitiNum** no implementa la fachada GCTMgr_I. Si bien se dice en el estándar que es obligatoria, no era de real interés para este proyecto. También hay que mencionar que la fachada MEMgr_I no es implementada por **mitiNum** sino por el Proyecto Gesinv. No lo discriminamos todavía para dejar en claro la interface. Este punto es retomado en la sección 6. *Interacción entre proyectos*, en la que se explica claramente la interacción con Gesinv y las interfaces definidas para ello.

Pasando a la comunicación entre capa de red y capa de servicio, **mitiNum** utiliza una interface definida por el grupo de proyecto, basada en el estándar CaSMIM especificado en el documento TMF508V30_pe.pdf. Esta interface fue adaptada para la comunicación de acuerdo a lo acordado en el alcance del proyecto, como se puede ver en la documentación.

Esta interface (como CaSMIM) comparte muchas de las ventajas de MTNM, en particular las que se desprenden de usar CORBA. Sin embargo, ambas interfaces (MTNM y CaSMIM) se basan en paradigmas distintos en cuanto a la granularidad de la interface. MTNM se basa en una granularidad gruesa “inteligente”, mientras que CaSMIM ofrece como alternativas granularidad gruesa por clase y granularidad fina (la opción obligatoria para CaSMIM es la granularidad gruesa por clase, y es la única soportada por miCaSMIM). Para una discusión más detallada de estos temas, puede verse la sección 5.5.1 *Granularidad*.

A continuación se presenta un diagrama con las interfaces exportadas por la capa de red.

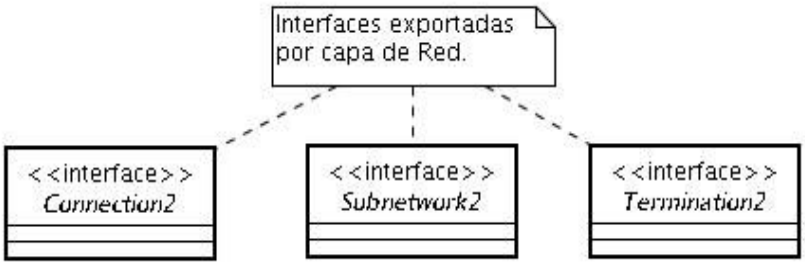


Diagrama 16 Interfaces de capa de red

5.4 Componentes

Los componentes implementados por el sistema se desprenden de la especificación de los estándares usados. En los Diagramas 17 y 18 se describen los componentes especificados por los estándares, los que son implementados por **mitiNum**, así como los que no, basados en las decisiones tomadas por el grupo de Proyecto teniendo en cuenta el alcance del mismo y las restricciones impuestas por los modelos usados.

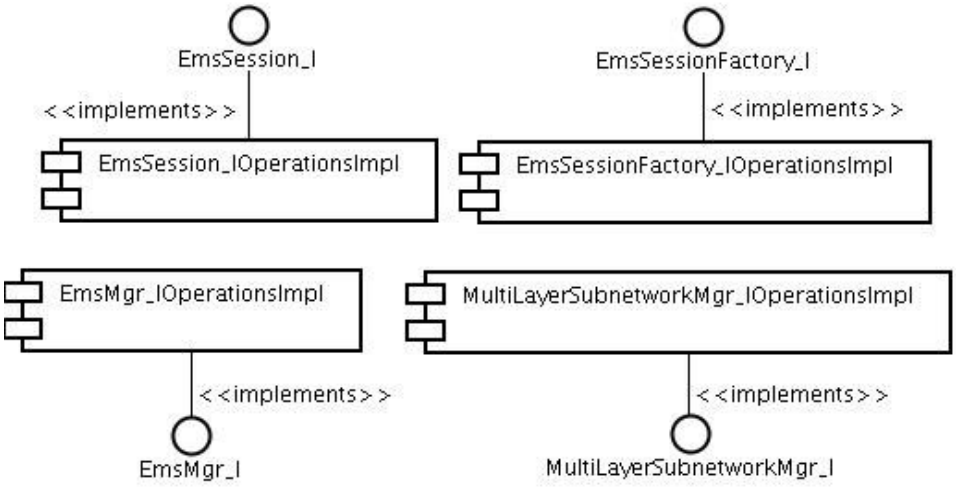


Diagrama 17 Componentes de capa de elemento

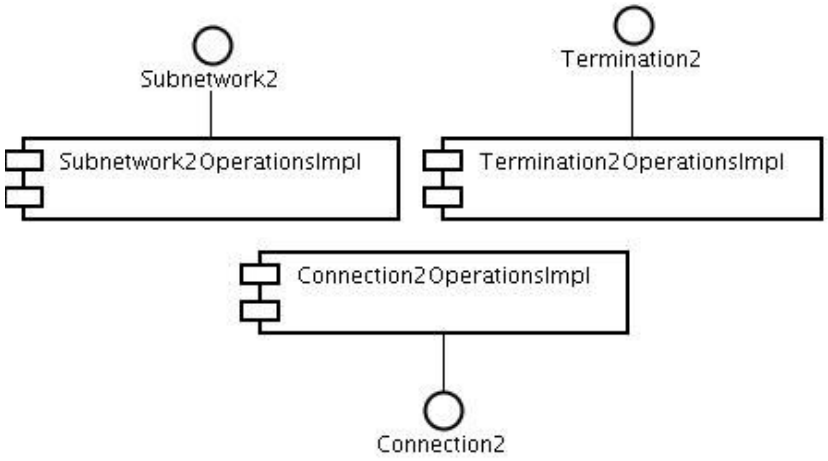


Diagrama 18 Componentes de capa de red

Estos diagramas intentan mostrar la arquitectura del sistema haciendo hincapié en la interacción entre capas, y por ello sólo se muestran interfaces y componentes involucrados directamente con su interacción.

5.5 Forma de usar CORBA

Veremos en este apartado algunos puntos sobre la forma en que CORBA es usado en el sistema.

Dos puntos a destacar son los referidos a la granularidad y a la forma de usar POA.

5.5.1 Granularidad

El concepto de granularidad se refiere a la forma de hacer corresponder los objetos de la interface a los objetos de la realidad modelada. Los dos enfoques básicos son granularidad fina y granularidad gruesa.

Granularidad fina

Hay una correspondencia uno-a-uno, donde a cada objeto del modelo le corresponde un objeto CORBA. Esta opción es la más fácil de diseñar, pero no necesariamente la más facial de implementar. En esta arquitectura los objetos de la aplicación residen directamente sobre la infraestructura CORBA, por lo que es la que ofrece la mayor flexibilidad a la hora de distribuir, y la más estrecha integración con los servicios de CORBA. A su vez, esta opción ofrece serios retos a la escalabilidad, ya que una aplicación en el área de las telecomunicaciones fácilmente puede llegar a tener millones de objetos.
El estándar CaSMIM ofrece una interface en esta modalidad (opcional), además de una versión de granularidad gruesa (obligatoria).

Granularidad gruesa

En esta arquitectura un objeto de la interface ofrece acceso a varios objetos del modelo, según algún criterio decidido durante el diseño. La forma más simple de esta modalidad es la *granularidad gruesa por clases*, donde cada objeto de la interface representa una clase del modelo. En esta arquitectura los objetos de la interface ofrecen los mismo métodos que la clase que representan, con un parámetro extra que es el identificador del objeto al que se le quiere aplicar la operación.

CaSMIM utiliza este enfoque para su interface en versión granularidad gruesa, donde las versiones de granularidad gruesa y fina difieren básicamente en la presencia de un parámetro identificador del objeto. Para minimizar las diferencias entre ambas opciones CaSMIM utiliza como identificador del objeto un nombre tal como lo utiliza el servicio de nombres de CORBA. De manera que al utilizar la opción de granularidad gruesa, uno provee el “nombre” como parámetro en el método del objeto que representa la clase correspondiente, mientras que en la opción de granularidad fina hay que recurrir al servicio de nombres de CORBA para recuperar el objeto CORBA que representa al objeto que se quiere acceder, e invocar el método sobre el.

La granularidad gruesa por clases es la arquitectura utilizada por la interface miCasmim, implementada para este proyecto.

Como alternativa a la granularidad gruesa por clases, se pueden agrupar objetos según otros criterios más inteligentes, por ejemplo teniendo en cuenta el nivel de acoplamiento entre los distintos objetos, los requerimientos reales de distribución, la cantidad esperada de instancias de objetos, su ciclos de vida, y como tal MTNM es un buen ejemplo de este enfoque. Los objetos de la interface MTNM (las fachadas) encapsulan objetos fuertemente acoplados según áreas funcionales. Esto tiene un gran impacto en la performance y en la claridad de las interfaces. La interface MTNM fue específicamente diseñada para ser fácil de usar desde la capa de red, lo que implica mucho más trabajo que diseñar una interface que sea únicamente *correcta*.

5.5.2 POATie.

Para crear un objeto CORBA hay que instanciar un objeto servant, asociarle una referencia (IOR) y por último activarlo. Un servant es un objeto Java [9] que implementa las operaciones de la interface.

El puente entre el objeto CORBA y el servant es POA (Portable Object Adapter). POA hace la correspondencia entre las solicitudes que llegan al servidor (en nuestro prototipo la clase Loader) y las instancias de objeto.

Existen dos formas de utilizar POA, por herencia de la clase *<interface>POA* o por delegación, usando la clase *<interface>POATie*.

En principio parece más sencillo y eficiente usar herencia ya que sólo hay un objeto Java por cada objeto CORBA. Sin embargo con la herencia de *<interface>POA* el objeto Java no puede heredar de ninguna otra clase y por lo tanto no puede implementar varias interfaces. El objeto Java no puede heredar de mas de una clase debido a que Java no soporta herencia múltiple.

El modo *Tie* permite que una clase de implementación pueda heredar de alguna otra clase Java y por lo tanto puede implementar varias interfaces a la vez y reutilizar otras clases.

Por lo tanto es recomendable utilizar delegación en lugar de herencia para implementar interfaces y fue la forma empleada en nuestro prototipo.

A continuación se presenta un ejemplo de implementación de una interface utilizando Tie.

Interface: `EmsSessionFactory_I`

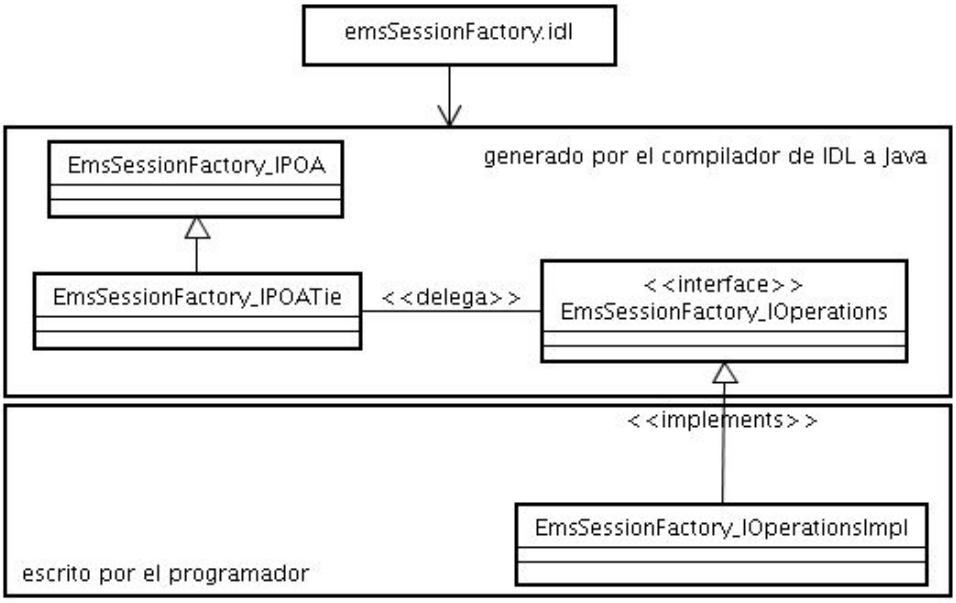


Diagrama 19 Implementación de una interface usando Tie

6 INTERACCION ENTRE PROYECTOS.

Abordaremos en este punto la interacción con el grupo de Proyecto Gesinv, e intentaremos dar una descripción de decisiones tomadas que impactan tanto en la arquitectura del sistema **mitiNum**, como en la arquitectura general.

Durante el transcurso de ambos proyectos, se dieron algunas dificultades para resolver algunos puntos del problema usando estrictamente los estándares usados. Esto motivó una serie de reuniones con el grupo Gesinv, y en la que se definieron una serie de interfaces extras a las mencionadas hasta ahora, interfaces totalmente internas a la capa para la que fue creada, y por tanto sin afectar a los estándares usados, o sea a las interfaces exportadas por cada una. Estos problemas en general estaban vinculados con el manejo de objetos que eran usados por ambos grupos, ya que los estándares no especificaban nada acerca de ello.

Cabe mencionar que esta interacción, independientemente de la capa en la que se estaba trabajando, debía mantener la ideología de trabajo en la que se enmarcaban ambos proyectos, y por tanto ciertas “reglas” que se debían cumplir.

En cuanto a los estándares se trata de no modificar sus interfaces, a menos que fuera estrictamente necesario, como el caso de CaSMIM, que fue modificado por razones de alcance del proyecto, pero que mantiene su base de definición.

En cuanto a la interacción de software, se debía usar CORBA por todas las cosas ya mencionadas, pero especialmente por las facilidades que ofrece en el desarrollo. También se consideró importante no agregar nuevas tecnologías a menos que fuera estrictamente necesario, para mantener el sistema tan simple como fuera posible, y no comprometer vías posteriores de desarrollo.

6.1 Capa de Elemento.

La primer etapa de ambos Proyectos fue decidir las responsabilidades de cada grupo en esta capa, ya que se debía (y quería) tener una separación clara. La decisión tomada se presenta en el siguiente diagrama.

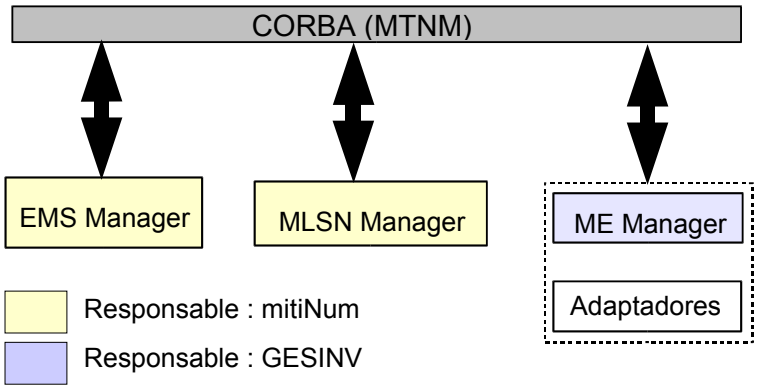


Diagrama 20 División en capa de elemento

En esta división la comunicación entre ambos grupos será usando MTNM, pero no será suficiente. Se definió además una interface auxiliar para el manejo de los objetos MTNM CrossConnects. Esto surge de la necesidad de realizar operaciones sobre ellos por parte de las fachadas que son responsabilidad de **mitiNum**, pero que para ello deben recurrir al componente MEManager que el es componente encargado de administrarlos. Nótese que la necesidad de estas interfaces extra es intrínseca a la arquitectura MTNM, y no al hecho que los componentes estaban siendo desarrollados por grupos de trabajo distintos.

Además de la interface antes mencionada, se crearon dos más, con propósito de una posible extensión del sistema. Estas interfaces permitirían crear y eliminar ciertos objetos para los cuales MTNM solo especifica los mecanismos de consulta, y que pueden ser de utilidad por ejemplo para que un posible componente de Discovery pueda agregar, modificar y eliminar estos objetos dinámicamente. Si bien esta interface no es estrictamente necesaria para la interacción ente ambos proyectos, su decisión fue tomada en conjunto. El siguiente diagrama resume lo explicado anteriormente.

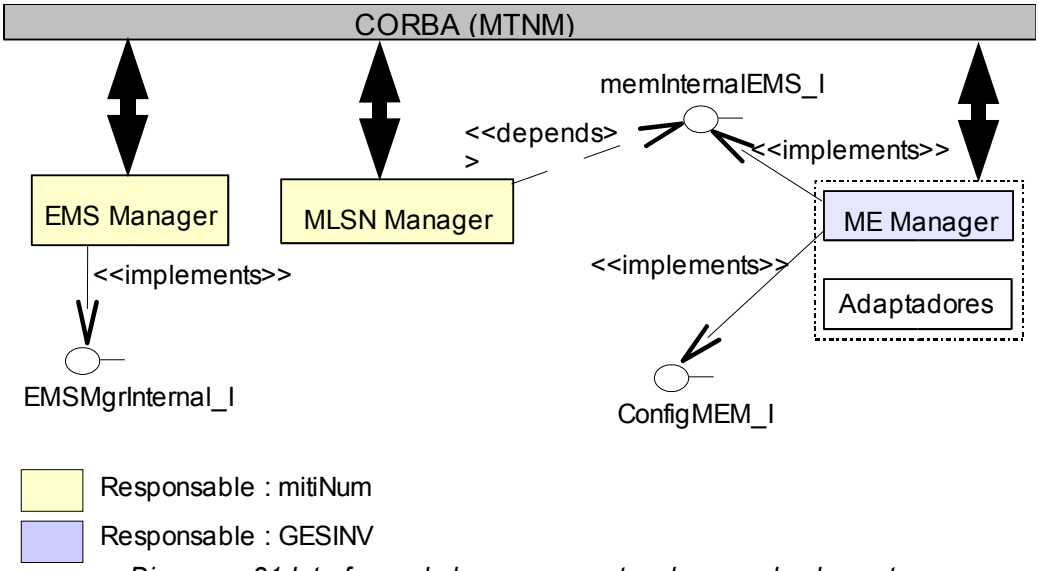


Diagrama 21 Interfaces de los componentes de capa de elemento

6.2 Capa de Red

En capa de red la interacción entre ambos proyectos fue un poco diferente a lo visto en capa de elemento, en donde las interfaces eran mas completas. Además en capa de red se tuvo mas libertad de elección, lo que llevó a usar distintos estándares. El grupo Gesinv se inclinó por el uso de WINMAN, mientras que **mitiNum** manteniendo la misma línea, decide usar una adaptación de CaSMIM[19].

En esta vista de la división entre proyectos, vemos la clara analogía con la división hecha para la capa de elemento, en donde por un lado tenemos el inventario y por el otro las funcionalidades de aprovisionamiento que trabajan sobre el inventario. Cabe recordar que esta división de procesos viene del TOM, como ha sido explicado antes.

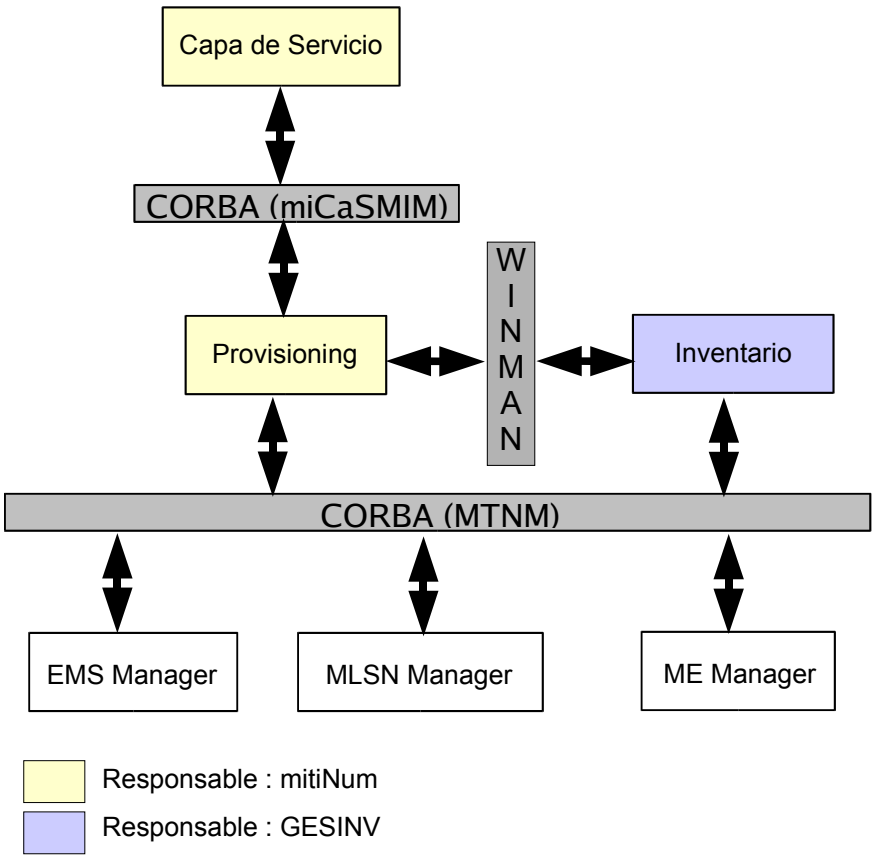


Diagrama 22 División en capa de red

19 Este punto ha sido desarrollado en el Capitulo 4.

Llendo mas al detalle de los componentes mencionados, intentaremos dar una visión mas terminada de los componentes involucrados en capa de red. La idea en la capa de red es que los puntos de acceso de la capa de servicio sean a través de Provisioning, siendo el Inventario interno a la capa. Con este esquema, la capa de servicio conoce a la capa de red por las interfaces que ofrece Provisioning, o sea miCasmim. A su vez Provisioning usa el Inventario vía WINMAN, que es lo usado por Gesinv. Con lo anterior tenemos que la interacción entre proyectos continúa manteniendo la idea general, que es el uso de estándares y la comunicación vía CORBA de ambos proyectos (como puede verse en el Diagrama 23), de manera de seguir manteniendo su independencia.

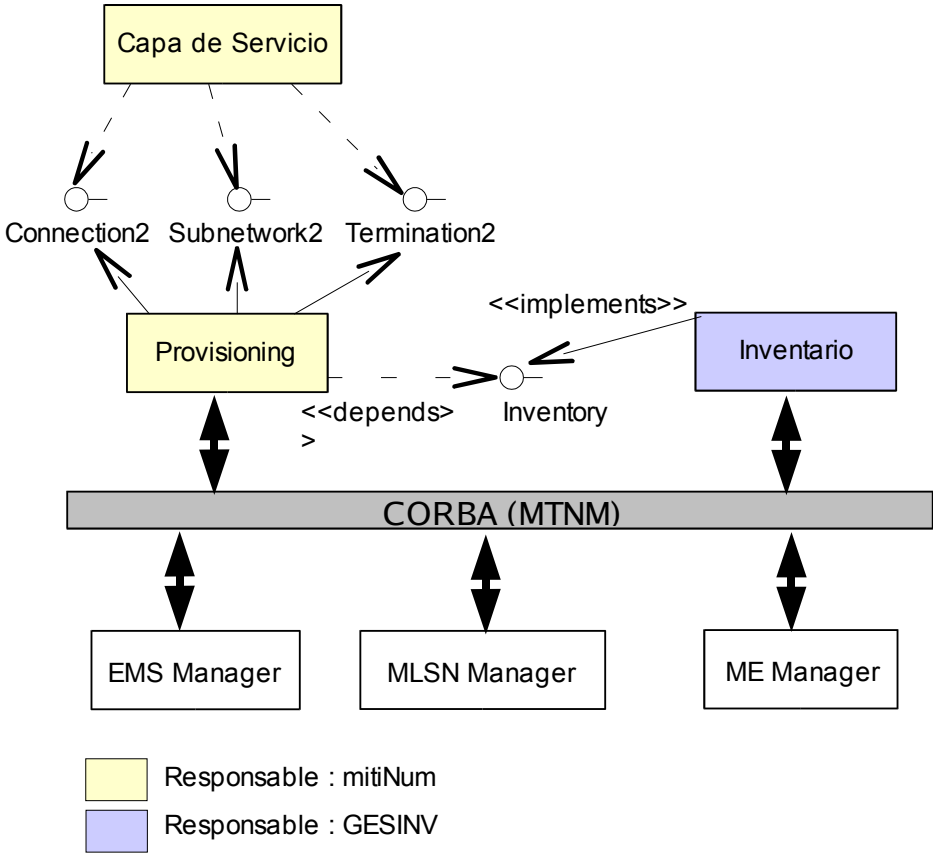


Diagrama 23 Interfaces de los componentes de capa de red

7 Selección de Plataforma

7.1 Lenguaje

Como lenguaje de desarrollo se selecciono Java [9].

Es de fundamental importancia para el impacto potencial de una herramienta en el mundo de las telecomunicaciones su capacidad para soportar un gran espectro de plataformas y entornos operativos, tanto de software (Linux, BSD, Solaris, Windows, etc) como de hardware (PC/Intel, Sparc, AS400, Alpha, etc.).

Java es orientado a objetos y se adapta naturalmente al desarrollo orientado a componentes.

Java provee muchas estructuras de datos potentes (listas, hashes) de forma fácil para usar, lo que simplifica mucho el prototipado.

Java está extremadamente difundido, lo que es una gran ventaja en cuanto a la disponibilidad de herramientas, documentación, y experiencia.

Java está disponible gratuitamente, y existen múltiples implementaciones ofrecidas por Sun, IBM, y versiones OpenSource. En este desarrollo se utilizó la versión de Sun.

7.2 Sistema Operativo

Para el desarrollo se utilizó Linux, por ser un sistema OpenSource y su gran penetración en el mundo de las telecomunicaciones. Las distribuciones usadas por los desarrolladores fueron RedHat8, 9 y SuSE9.

Sin embargo, se tuvo en cuenta que el sistema debía poder correr en múltiples plataformas, por lo que se testeo en entornos con Windows.

7.3 ORB

Las características críticas para la selección de un ORB [18] para este proyecto son:

- Sin problemas de licenciamiento (preferentemente OpenSource)
- Soportar Java.
- Servicio de nombres
- Servicio de notificaciones
- Multiplataforma.

ORB	*Lenguaje*	*Servicio de nombres*	*Servicio de notificaciones*	*Licenciamiento*
IBM Component Broker	C++, Java	Si	No	Comercial
CorbaPlus	C++, Java	Si	Si	Evaluación 60 días
GemORB	Java, Smalltalk	Si	No	?
Interstage	Java, C, C++ y COBOL	Si	Si	Comercial
JacORB	Java	Si	Si	OpenSource
JavaORB	Java	Si	No	OpenSource
Orbix	C++, Java, COBOL, PL/I	Si	Si	Evaluación gratis
VisiBroker	C++, Java, Delphi	Si	No	Evaluación gratis
Sun JavaIDL	Java	Si	No	Gratis (incluida con JDK)
MICO	C++, Eiffel	Si	Si	OpenSource
ORBacus	C++, java	Si	No	Gratis para uso no comercial

Tabla 1 Selección de ORB

Se seleccionó JacORB, por cumplir con todos los requisitos.

7.4 Otras Herramientas

7.4.1 IDE

Como entorno de desarrollo integrado se utilizó SunONE, debido a su disponibilidad, potencia, y su integración con CORBA y CVS.

7.4.2 Sistema de versionado

De los tres equipos utilizados en el desarrollo, dos usaban un repositorio de fuentes y documentación (CVS). Estos dos equipos estaban en la misma LAN.

7.4.3 Modelado UML

El modelado UML se realizó utilizando la herramienta JUDE[20]

7.4.4 Paquete de oficina

Se utilizó OpenOffice.

7.4.5 Librerías

Para acceder a archivo XML se utilizó la librería JAXB[19] de Sun[21].

20 Se puede obtener JUDE en el CD de instalación de mitiNum en herramientas/UML o en la URL sourceforge.net/projects/stjude

21 Ver Apéndice II: XML/XSD.

7.4.6 Base de Datos Relacional

Para instalar el componente de inventario (realizado por el grupo de proyecto Gesinv, ver sección *6.Interacción entre proyectos*) fue necesaria una RDBMS. Se utilizó Postgres [20], junto con su driver JDBC.

8 Capa de Elemento

8.1 Inicialización y configuración

La naturaleza distribuida y orientada a componentes del sistema plantea requerimientos especiales para la puesta en marcha y configuración. Para simplificar estas tareas, se decidió implementar una infraestructura que permitiera:

- Configurar el deployment del sistema en archivos de configuración
- Gestionar la configuración de los componentes de forma normalizada
- Centralizar las tareas relacionadas con la configuración de los servicios CORBA (servicio de nombres, canal de notificaciones)

A su vez, se decidió ignorar los problemas de inicialización y monitoreo remotos, y todo lo vinculado a la resistencia a fallos. Esto requiere un estudio más profundo de CORBA, que no aporta a los objetivos de este proyecto.

El resultado es el componente Loader, que recibe como parámetro un archivo XML. Este archivo está especificado usando XMLSchema, y es procesado usando JAXB[22]. El esquema es:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="configuration" type="ConfigurationType"/>

<xsd:complexType name="ConfigurationType">
  <xsd:sequence>
    <xsd:element name="channel" type="ChannelType" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="component" type="ComponentType"
                 minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="ns_prefix" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="ChannelType">
  <xsd:attribute name="ns_name" type="xsd:string"/>
  <xsd:attribute name="create" type="xsd:boolean"/>
</xsd:complexType>

<xsd:complexType name="ComponentType">
  <xsd:sequence>
    <xsd:element name="parameter" type="ParameterType"
                 minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="java_class" type="xsd:string"/>
  <xsd:attribute name="ns_name" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="ParameterType">
  <xsd:attribute name="name" type="xsd:string"/>
  <xsd:attribute name="value" type="xsd:string"/>
</xsd:complexType>

</xsd:schema>
```

22 Ver Apéndice II: XML/XSD.

Un archivo de configuración típico tiene la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration ns_prefix="proyGrado">
    <channel ns_name="canal_ems" create="true"></channel>
    <component java_class="EMSMgr" ns_name="EMS">
        <parameter name="archivoInit" value="mitiNum/ems/emsInit.xml"></parameter>
        <parameter name="intervaloPing" value="8000"></parameter>
    </component>
    <component java_class="MultiLayerSubnetworkMgr" ns_name="MultiLayerSubnetwork">
        <parameter name="archivoInit" value="mitiNum/ems/sncInit.xml"></parameter>
        <parameter name="intervaloPing" value="8000"></parameter>
        <parameter name="maxIteradores" value="50"></parameter>
        <parameter name="iteradoresTimeout" value="3600000"></parameter>
    </component>
</configuration>
```

El Loader al procesar este archivo realiza las siguientes acciones:

- Crea un canal de notificaciones, y lo registra en el servicio de nombres como “canal_ems.proyGrado”
- Instancia el componente EMSMgr, y lo registra con el nombre “EMS.proyGrado”...
- ...y le provee de parámetros.
- Instancia el componente MultiLayerSubnetworkMgr, y lo registra con el nombre “MultiLayerSubnetwork.proyGrado”...
- ...y le provee de parámetros.

Esto permite especificar varias configuraciones interesantes, modificando únicamente los archivos de configuración. Por ejemplo, los siguientes archivos especifican una arquitectura donde parte de los componentes se encuentran en un equipo y el resto en otro.

```
<configuration ns_prefix="proyGrado">
    <channel ns_name="canal_ems" create="true"></channel>
    <component java_class="EMSMgr" ns_name="EMS">
        <parameter name="archivoInit" value="mitiNum/ems/emsInit.xml"></parameter>
    </component>
    <component java_class="MultiLayerSubnetworkMgr" ns_name="MultiLayerSubnetwork">
        <parameter name="archivoInit" value="mitiNum/ems/sncInit.xml"></parameter>
    </component>
</configuration>
```

```
<configuration ns_prefix="proyGrado">
    <channel ns_name="canal_ems" create="false"></channel>
    <component java_class="EmsSessionFactory" ns_name="EmsSessionFactory">
        <parameter name="mtnm_version" value="2.0"></parameter>
        <parameter name="EMS" value="EMS.proyGrado"></parameter>
        <parameter name="MultiLayerSubnetwork"
                   value="MultiLayerSubnetwork.proyGrado"></parameter>
    </component>
</configuration>
```

Nótese la línea `<channel ns_name="canal_ems" create="false">` en el segundo archivo. Con false se indica que no se desea crear un nuevo canal de notificaciones, si no acceder a uno ya creado (y registrado como canal_ems.proyGrado). Esto permite que varios componentes físicamente distribuidos utilicen el mismo canal de notificaciones.

La única limitación para distribuir en múltiples equipos proviene de las limitaciones internas de los componentes, en cuanto a sus interfaces requeridas, y al momento en que intentan accederlas. En el ejemplo anterior, si se quiere levantar el primer archivo, es necesario que pueda levantarse los componentes EMSMgr y MultiLayerSubnetwork mientras el componente EMSSessionFactory aún no está disponible. Debe notarse que el archivo que instancia el canal de notificaciones debe ser ejecutado antes que todos los que quieran usarlo.

Al utilizar varios archivos también se puede crear varios canales, de forma que los componentes inicializados en cada archivo quedan aislados entre ellos.

Los parámetros pasados a los componentes consisten en pares atributo-valor. Esto ofrece la máxima flexibilidad. Cada componente debe documentar qué atributos utiliza.

8.1.1 Detalles de implementación.

Para permitir que un nuevo componente sea gestionado por el Loader, se deben realizar dos tareas:

1. La clase que representa al componente debe tener un constructor en el formato especificado a continuación.
2. Se debe agregar código al Loader para soportar el componente dado.

Constructor de la Clase.

La clase debe tener un constructor con la firma:

```
public ComponenteImpl(org.omg.CORBA.ORB orb, org.omg.PortableServer.POA poa,
EventChannel canal, Map parametros)
```

Los parámetros del constructor son:

1. orb: El ORB a ser usado por este componente (es compartido con todos los componentes levantados por el Loader)
2. poa: El POA es compartido por todos los componentes levantados por el Loader, y al momento de la invocación ya está activo. Esto implica que pueden empezar a llegar solicitudes de servicio tan pronto como el componente se registra. Esto debe ser tenido en cuenta para determinar el orden en que se levantan los componentes.
3. canal: El canal de notificaciones. Este es el canal usado para las notificaciones especificadas por MTNM.
4. parametros: En esta estructura llegan los parámetros especificados en el archivo xml de inicio. Cada parámetro es un par de strings, donde el *key* del Map es el nombre del parámetro, y el *value* es el valor.

Extender Loader

Para que el Loader soporte un componente, debe existir una entrada correspondiente del método:

```
private static CargarComponente(ORB orb, POA poa, EventChannel canal, String clase, Map
parametros)
```

Los parámetros `orb`, `poa`, `canal`, y `parametros` son para ser pasados al constructor del componente. El parámetro clase contiene una etiqueta que identifica el componente. Esta etiqueta es la misma que se extrae del archivo xml de inicio, del atributo *java_class* del nodo *component*. En la Tabla 2 se presentan las etiquetas soportadas por el Loader al momento.

Etiqueta	*Clase asociada*
EmsSessionFactory	mitiNum.ems.mtnmImpl.EmsSessionFactory_IOperationsImpl
MultiLayerSubnetworkMgr	mitiNum.ems.mtnmImpl.MultiLayerSubnetworkMgr_IOperationsImpl
EMSMgr	mitiNum.ems.mtnmImpl.EMSMgr_IOperationsImpl
Subnetwork2	mitiNum.nms.provisioning.miCasmimImpl.Subnetwork2OperationsImpl
Connection2	mitiNum.nms.provisioning.miCasmimImpl.Connection2OperationsImpl
Termination2	mitiNum.nms.provisioning.miCasmimImpl.Termination2OperationsImpl
Routing	mitiNum.nms.provisioning.miCasmimImpl.Routing2OperationsImpl
ManagedElementMgr	gesinv.ems.mem.MEM

Tabla 2 Loader - etiquetas soportadas

Para que el método `CargarComponente` soporte la nueva etiqueta, debe agregarse un caso al *if...else*. La entrada típica es como sigue:

```
...
 } else if (clase.equals("etiqueta")) {
       //1) Instanciar y registrar el componente.
       //2) Asignar la fachada a MgrImplObj
 } else ...
```

El paso 1) consiste de instanciar el objeto que levanta el componente, pasándole los parámetros `orb`, `poa`, `canal` y `parametros`, y dejarlo activo y disponible. Nótese que puede usarse el método auxiliar `InstanciarComponente(...)`.

El objetivo del paso 2) es indicar que objeto CORBA se desea se registre en el servicio de nombres con el nombre indicado en el archivo xml de inicio en el atributo `ns_name` del nodo `component`. Si la variable `MgrImplObj` no se asigna, Loader no registrará nada en el servicio de nombres.

Nótese que con esta estructura, se puede escribir soporte para un componente que use una clase *bootstrap* que levante potencialmente varios componentes, configure el servicio de nombres ella misma sin recurrir al Loader, y que puede tener un constructor de forma arbitraria. Un ejemplo puede verse en la etiqueta *ManagedElementMgr*, que corresponde a un componente desarrollado por el grupo de proyecto “Gestión de Inventario”.

8.2 Sesiones

La gestión de sesiones está especificada en el documento TMF513, en los casos de uso 5.2 NMS-EMS Session Management Use Cases, y en el documento TMF608, diagrama 6.1.13 SessionFactory Class Diagram y el punto 6.1.21.40 al 6.1.21.43 del diccionario de clases.

Las sesiones permiten tener un punto de acceso único a los recursos de capa de elemento, de forma de poder realizar tareas de control de acceso, y de control de utilización de recursos. También ofrece mecanismos para detectar fallos de comunicación entre componentes.

Una sesión abierta es representada por un par de instancias de componentes, uno a cargo del cliente (NmsSession) y otro a cargo de la capa de elemento (EmsSession). Cada uno de estos componentes posee una referencia al otro. Esta referencia es usada para verificar el estado de la conexión.

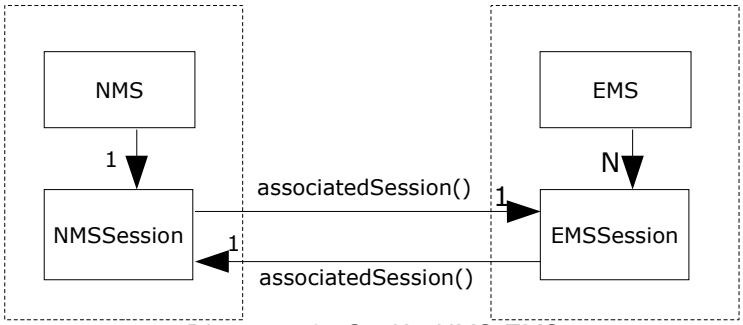


Diagrama 24 Sesión NMS-EMS

El EMS mantiene una instancia de EmsSession por sesión abierta. El EMS periódicamente invoca `emsSession.asociatedSession().ping()`. Si la invocación falla, esto indica que la conexión se perdió (ya sea por problemas de conectividad o por que el cliente está inestable), por lo que el EMS considera la comunicación terminada y cierra la sesión, destruyendo el objeto emsSession y liberando todos los recursos asociados. El NMS debería hacer lo mismo.

La cantidad máxima de sesiones y los tiempos entre verificaciones son configurables[23].

Un cierre de sesión controlado por parte del NMS consiste en la invocación de `endSession()` sobre el componente de sesión que se posee. La forma de informar del cierre al componente asociado y la implementación de la lógica necesaria debe estar encapsulada en los componentes de sesión y el EMS.

El componente encargado de generar los objetos EmsSession es la EmsSessionFactory, por el método `getEmsSession(...)`. La fachada EmsSessionFactory junto con los componentes EMSSession y NMSSession componen lo que se conoce como *root interface*. Este subsistema permite acceso a los recursos de la EMS, a saber:

- Información sobre qué fachadas están implementadas.
- Las fachadas que implementan las interfaces de MTNM
- El canal de eventos, que permite recibir mensajes asincrónicos generados en el EMS
- Información sobre la versión del protocolo implementado

23 Ver sección 8.2.5 Detalles de implementación.

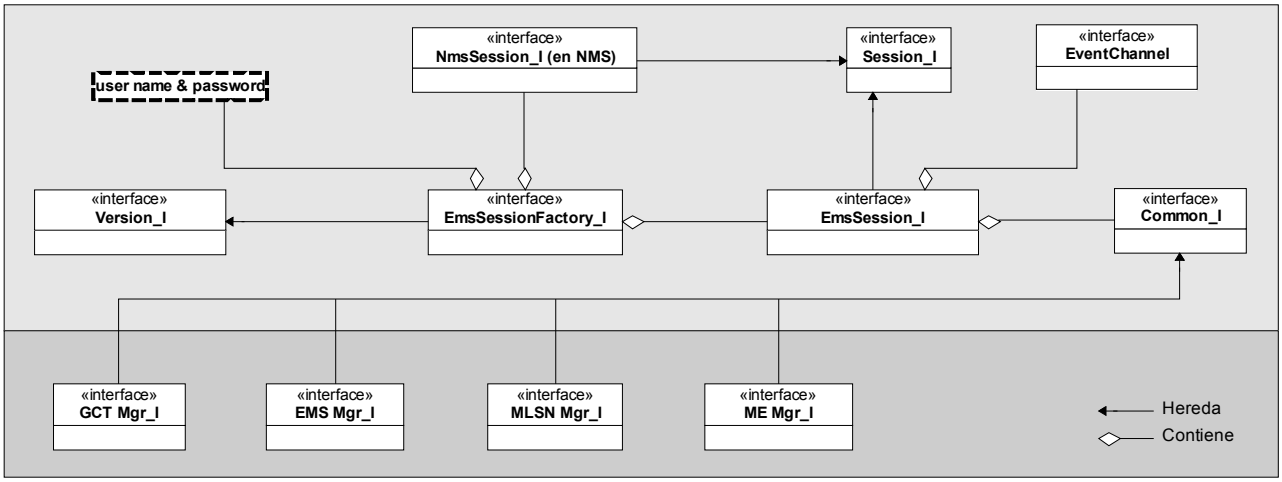


Diagrama 25 Root interface

El ciclo de vida típico de una sesión iniciada por el NMS se muestra a continuación.

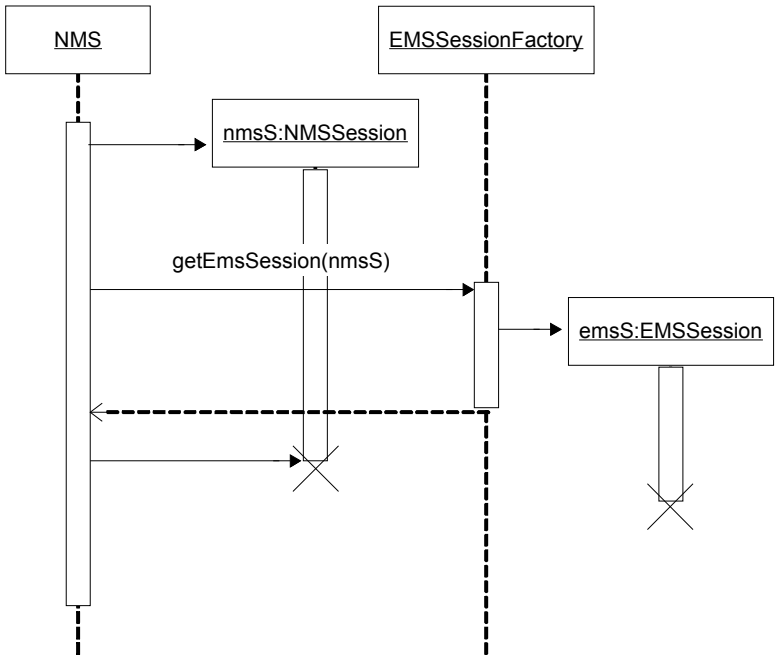


Diagrama 26 Ciclo de vida de una sesión

Debe notarse que no está especificado como se notifica del cierre de sesión al EMS. Esto queda libre a la implementación. Tampoco está especificado cómo las distintas fachadas obtienen información de las sesiones disponibles o sobre la identidad de la sesión que está accediendo, ya sea para realizar control de acceso o para poder rastrear la utilización recursos.

Al abrirse una conexión, el EMSSessionFactory crea una objeto EMSSession, con una referencia al NMSSession del cliente. Este NMSSession identifica que el cliente se ha validado correctamente ante el EMS. Ahora, un acceso cualquiera a una fachada MTNM, por ejemplo, el método `setNativeEMSName` de la fachada MultiLayerSubnetworkMgr, no identifica al cliente: no hay una manera obvia de asociar al origen de la invocación con alguna de las sesiones abiertas en el EMSSessionFactory. Por lo tanto, las fachadas no tienen la información suficiente para verificar si la invocación proviene de un cliente validado.

La forma natural de resolver el problema sería que los métodos MTNM recibieran siempre un parámetro extra, consistente de un identificador de sesión. Sin embargo, la especificación MTNM no preve tal opción.

Asimismo, como resultado de una invocación pueden instanciarse recursos (típicamente iteradores) que están asociados a la sesión que los generó[24]. Si la sesión que solicitó un iterador se cierra, este iterador debería ser liberado. Para esto, debe mantenerse información que permita identificar las sesiones responsables de los recursos. A su vez, al ser el iterador un componente CORBA debería poder controlarse que el acceso al mismo sea realizado por el mismo cliente que lo generó (y que fue validado como con permisos para generarlo). Este es un problema interesante para resolver, y que está íntimamente vinculado a la seguridad del sistema[25].

8.2.1 Alcance

Del subsistema de gestión de sesiones se consideró importante implementar lo siguiente:

- EMSSessionFactory:
  - Generación y destrucción de objetos EMSSession, de acuerdo al estándar MTNM.
  - Monitoreo del estado de conexión con el cliente de capa de red
  - Capacidad de configurar la cantidad máxima de sesiones admitidas.
  - Capacidad de configurar la frecuencia de verificaciones del estado de la conexión
  - Devolver referencias IOR correctas a las fachadas en el método `getManager()`
  - `getVersion()`
- EMSSession
  - Devolver el canal de eventos utilizado por el EMS con el método `getEventChannel`()
  - Devolver la lista de fachadas disponibles en `getSupportedManagers()`
  - Cierre controlado de la sesión con `endSession()`
  - `getAssociatedSession()`
- NMSSession
  - Cierre controlado de la sesión con `endSession()`
  - `getAssociatedSession()`

No se implementó las siguiente funcionalidades:

- EMSSessionFactory:
  - Sistema de control de acceso. Únicamente se implementó una solución trivial a la gestión de usuarios/contraseñas. Las fachadas asumen que los accesos provienen de clientes validados.
  - Monitoreo de recursos asignados a los usuario (el riesgo de agotamiento de recursos se resolvió de otra manera: ver sección *8.3 Iteradores*)
- NMSSession
  - Mecanismos de control de pérdidas de eventos en el canal de notificaciones (métodos `eventLossOccurred()` y `eventLossCleared()`)

24 Ver sección 8.3 Iteradores.

25 Ver sección 8.8 Temas no especificados.

8.2.2 Inicialización

Para obtener la información de configuración necesaria para su funcionamiento, el subsistema de sesiones utiliza el mecanismo común, implementado en la clase Loader[26]. El fragmento correspondiente del archivo de inicialización se presenta a continuación:

```
<component java_class="EmsSessionFactory" ns_name="EmsSessionFactory">
    <parameter name="intervaloPing" value="8000"></parameter>
    <parameter name="mtnm_versión" value="2.0"></parameter>
    <parameter name="EMS" value="EMS.proyGrado"></parameter>
    <parameter name="MultiLayerSubnetwork"
               value="MultiLayerSubnetwork.proyGrado"></parameter>
    <parameter name="ManagedElement" value="ManagedElement.proyGrado"></parameter>
</component>
```

Los parámetros usados son los siguientes:

Parámetro	***Tipo***	***Valor por defecto***	***Descripción***
java_class	String		Es una etiqueta que indica al Loader la identidad del componente a levantar (en este caso, el EmsSessionFactory)
ns_name	String		El nombre con el que se quiere que se registre el componente en el servicio de nombres de CORBA. El EmSessionFactory es el único componente al que los clientes de capa de red deben acceder usando el servicio de nombres.
intervaloPing	Integer	30000	Cada cuantos milisegundos se verifican las sesiones. Debido a detalles de implementación, las sesiones se cierran únicamente durante un período de verificación, aunque hayan sido cerradas de forma controlada (método *lazy*)
mtnm_versión	String	2.0	Versión que va a devolver `getVersion()`
EMS	String		Nombre en el servicio de nombres de CORBA de la fachada EMSMgr. Este nombre es usado por el EMSSession para resolver la llamada a `getManager()`
MultiLayerSubnetwork	String		Nombre en el servicio de nombres de CORBA de la fachada MLSNMgr. Este nombre es usado por el EMSSession para resolver la llamada a `getManager()`
ManagedElement	String		Nombre en el servicio de nombres de CORBA de la fachada MEMgr. Este nombre es usado por el EMSSession para resolver la llamada a `getManager()`

Tabla 3 Parámetros del EmsSessionFactory

26 Ver sección 8.1 Inicialización y configuración.

8.2.3 Persistencia

Debido a que al especificar el alcance se omitió la necesidad de mantener información sobre asignación de recursos a sesiones, no hace falta mantener ninguna información de forma persistente.

8.2.4 Interfaces internas

Con las limitaciones especificadas en el alcance, no hay necesidad de agregar interfaces extra a las ya especificadas en MTNM

8.2.5 Detalles de implementación.

Sesiones

El esquema implementado para gestionar las sesiones se resume en lo siguiente:

- Un cierre controlado de sesión (invocación de `endSession()`) causa que la instancia del componente de sesión afectada se desactive y destruya.
- El EMSSessionFactory considera que si una invocación `emsSession.asociatedSession().ping()` sobre un objeto de EMSSession falla (genera una excepción), esto indica que la sesión se cortó por problemas de comunicación, o ha sido cerrada de forma normal por el cliente de capa de red. En este caso procede a invocar `emsSession.endSession()`, lo que destruirá su mitad de la conexión.
- Las sesiones se revisan en forma periódica.

En otras palabras, los casos de cierre de sesión accidental y controlado se manejan de la misma manera, donde el caso de cierre controlado equivale a una verificación *lazy*.

EMSSessionFactory

La clase EMSSessionFactory mantiene una lista de sesiones abiertas. Esta lista está sincronizada, para permitir el acceso concurrente a los métodos de la fachada.

```
private List _sesiones=Collections.synchronizedList(new LinkedList());
```

Cada item de la lista es el del tipo EmsSession_I, o sea que es el IOR de un objeto CORBA. Durante la inicialización de la clase, se programa un timer, que invocará el método privado `ChequearConexion()` cada cierto lapso de tiempo pre-configurado[27]. Este método recorre la lista de sesiones, realizando el control del estado de la conexión y destruyendo objetos EMSSession cuando hiciere falta.

Es importante tener en cuenta los potenciales problemas de sincronización. Si durante un recorrido de la lista de sesiones llegara una invocación de `getEMsSession()`, esta intentará insertar sobre esa misma lista.

Otro problema potencial ocurre debido a que un cliente puede invocar `getEMSSession()` cuando su POA está aun inactivo[28]. Esto causa un *deadlock* si EMSSessionFactory intenta invocar `ping()` sobre el nmsSession de este cliente: el cliente está bloqueado esperando `getEMsSession()`, y el EMSSessionFactory está bloqueado esperando el `ping()`. Para este problema, se identificaron las siguientes soluciones, cada una de ellas suficiente:

- Los clientes no intentan abrir sesiones hasta que su POA se activa. No es robusta, ya que depende de los detalles de implementación del cliente. No se implementó.

27 Ver sección 8.1 Inicialización y configuración.

28 Ver sección 5.5 Forma de usar CORBA.

- El POA usado por el servidor se activa antes de levantar el componente. Debe tenerse cuidado con el orden en que se levantan los componentes en el POA activado, y con la posibilidad de intentar acceder a un componente que aún no se levantó. Se implementó.
- El método `ChequearConexion()` ignora las sesiones cuyos POA están inactivos. Hace falta una extensión de la interface. No se implementó.

Para recorrer la lista de sesiones, `ChequearConexion()` no la bloquea durante todo el tiempo de la recorrida, sino que únicamente mientras analiza un item. Esto se hace pare reducir los tiempos de respuesta para `getEmsSession()` en caso de que llegue durante un período de revisión.

```
    private void ChequearConexiones() {

        for (int i=0;i<_sesiones.size();i++){
            synchronized(_sesiones) {
                EmsSession_I sesion=(EmsSession_I)_sesiones.get(i);
                try {
                    Session_I sessionAsoc=sesion.associatedSession();
                    sessionAsoc.ping();
                } catch (Exception ex) {
                    try {
                        sesion.endSession();
                    } catch (Exception e) {
                    }
                    _sesiones.remove(i);
                }
            }
        }

    }
```

El método `ChequearConexiones()` se invoca automáticamente por un timer inicializado por el siguiente código:

```
int intervaloPing=Integer.parseInt(_parametros.get("intervaloPing").toString());
new Timer(true).scheduleAtFixedRate(new TimerTask() {
    public void run() {
        ChequearConexiones();
    }
}, intervaloPing, intervaloPing);
```

La frecuencia de revisiones es un compromiso entre el tráfico de red necesario para contactar cada sesión, y el retraso en el efectivamente liberar las sesiones cerradas. Para un sistema que soporte gran cantidad de sesiones, o sesiones que se crean y destruyen con mucha frecuencia, convendría implementar otra política donde en vez de invocar `ChequearConexion()` a intervalos fijos, este se reprograma para un lapso de tiempo dado luego de terminar su ejecución (esto es, en vez de invocarse cada N segundos, se invoca N segundos después de haber terminado). En la práctica, esto se reduce a utilizar `schedule(...)` en vez de `scheduleAtFixedRate(...)` en el código anterior.

EMSSession

El método `getManager(...)` busca las IOR de las fachadas en el servicio de nombres, según los nombres registrados en la configuración.

8.3 Iteradores

Para posibilitar el intercambio de grandes cantidades de información, MTNM recurre al patrón de diseño *iterador*. Los iteradores son usados por todos los métodos MTNM que tienen que devolver una cantidad no acotada de objetos. Los métodos MTNM son típicamente de la forma:

```
getAllObjectName(
       (in qualifiers) /* some operations may supply qualifiers */
       in unsigned long how_many,
       out objectList list,
       out iterator iteratorReference)
raises(globaldefs::ProcessingFailureException);
```

Un iterador se implementa como un componente CORBA, que ofrece una interface de la forma:

```
interface TipoIterator_I
   {
       boolean  next_n(in unsigned long how_many, out TipoList_T  objList)
                raises (globaldefs::ProcessingFailureException);
       unsigned long getLength()
                raises (globaldefs::ProcessingFailureException);
       void     destroy()
                raises (globaldefs::ProcessingFailureException);
   };
```

La forma de usar iteradores está descrita en el documento `IDL_V2.0/supportingDocumentation/iterators.html`.

Además de las interfaces especificadas por MTNM, que sirven para consultar información de los iteradores, hacen falta interfaces que especifiquen cómo se cargan los datos en el iterador.

8.3.1 Alcance

Debido a que los objetivos del prototipo no implicaban grandes requerimientos de escalabilidad, se consideró suficiente que la lista de items a ser manejada por un iterador fuera pasada al mismo en el constructor, en una operación atómica. En una implementación *"enterprise level"*, se debería permitir agregar items luego de instanciado el iterador. Hay que tener en cuenta que la especificación MTNM 2.0 (la usada en nuestro prototipo) tiene algunas limitaciones en este escenario, que fueron corregidas en la versión 2.1. En particular, se especificó el comportamiento en el caso que el usuario de un iterador intenta sacar información más rápidamente de lo que esta se carga.

Como se menciona en la sección dedicada a las sesiones, los iteradores deberían estar asociados a sesiones abiertas, de forma de poder liberarlos en caso de cierre de sesión (controlado o por pérdida de comunicación). Debido a que esta funcionalidad se omitió del alcance de las sesiones, para evitar fugas de memoria se recurrió a *timeouts* que aseguren que no queden iteradores "huérfanos"

Para el escenario indicado se hace un manejo correcto del ciclo de vida de los iteradores: se liberan los recursos de iteradores una vez "vaciados" por sus usuarios, y o una vez vencido su timeout; se puede limitar la cantidad máxima de iteradores abiertos.

8.3.2 Inicialización

Las fachadas que usan iteradores reciben parámetros que configuran su funcionamiento desde el archivo común de inicialización[29].

Los parámetros definidos son:

Parámetro	*Tipo*	*Valor por defecto*	*Descripción*
intervaloPing	Integer	8000	Cada cuantos milisegundos se verifican los iteradores. Debido a detalles de implementación, los iteradores se destruyen únicamente durante un período de verificación (método *lazy*), en caso de nos ser destruidos explícitamente por su cliente.
maxIteradores	Integer	1000	Cantidad máxima de iteradores abiertos simuláneamente. En caso de superar esta cantidad, se devuelve la excepción `EXCPT_TOO_MANY_OPEN_ITERATORS`. Un maxIteradores igual a 0 indica que no hay un tope máximo.
iteradoresTimeout	Integer	28800000	Tiempo de vida máximo de un iterador. Luego de transcurrido este tiempo, el iterador es destruido y liberado.

Tabla 4 Parámetros relacionados con Iteradores

8.3.3 Persistencia

Debido a su naturaleza, los iteradores no deben mantener información entre instancias de su componente creador, por lo que no hay requerimientos de persistencia.

8.3.4 Interfaces internas

Con las limitaciones especificadas en el alcance, no hay necesidad de agregar interfaces extra a las ya especificadas en MTNM.

.

8.3.5 Detalles de implementación.

Cada fachada que ofrece iteradores como mecanismo para devolver información, mantiene listas de iteradores, una por cada tipo de iterador. El ejemplo siguiente es extraído de la clase `EMSMgr_IOperationsImpl`, que implementa la fachada EMSMgr.

```
...
private List _snIters=Collections.synchronizedList(new LinkedList());
private List _snNameIters=Collections.synchronizedList(new LinkedList());
private List _tlIters=Collections.synchronizedList(new LinkedList());
private List _tlNameIters=Collections.synchronizedList(new LinkedList());
...
```

29 Ver sección 8.1 Inicialización y configuración.

Cada lista contiene elementos de la clase `iterador.RegIterator_T`. Esta es una clase abstracta que permite homogeneizar las tareas de administración de los iteradores, haciéndola independiente del tipo particular de iterador. Esto permite tener una sola lista de iteradores, en vez de cuatro como en el ejemplo, pero mantenerlas separadas permite simplificar el control de cantidad máxima de iteradores, que está definida por tipo.

Esta clase es utilizada por las fachadas para realizar la administración de los iteradores. Cada cierto tiempo[30] las fachadas recorren la lista de iteradores. Sobre cada lista de iteadores se ejecuta el siguiente código:

```
while (its.hasNext()) {
    RegIterator_T regIterator=(RegIterator_T)its.next();
    if ( regIterator.comoParaBorrar(_parametros) || !regIterator.hayMas() ) {
        regIterator.Eliminar();
        its.remove();
    }
}
```

Hay 3 causas para que un iterador sea destruido y liberado:

1. El iterador ha sido destruido explícitamente por su cliente, mediante la invocación al método `destroy()` del iterador. En este caso, el método `hayMas()` detecta un fallo de comunicación con el iterador, y devuelve *false*.
2. El iterador es vaciado por su cliente, pero no lo cierra explícitamente. El método `hayMas()` detecta esta condición invocando `iter.next_n(0,new SubnetworkList_THolder())`, de forma de verificar si quedan elementos, pero sin extraer ningún elemento.
3. Un iterador excede su tiempo de vida[31]. La clase `RegIterator_T` mantiene un atributo donde registra el tiempo de creación del objeto. Este atributo (junto con los parámetros de la fachada) es usado por el método `comoParaBorrar(...)` para determinar si el *timeout* ha pasado.

30 Ver sección 8.3.2 Inicialización.

31 Ver sección 8.3.2 Inicialización.

8.4 Clases utilitarias

Las clases utilitarias se crearon con el objetivo de ser usadas por todos los componentes de manera de unificar conceptos y evitar el re-trabajo.

8.4.1 NombreMtnm

La clase `NombreMtnm.java` incluida en el paquete `mitiNum.ems.mtnmImpl` tiene en cuenta la especificación dada en `globaldefs::NameAndStringValue_T` de MTNM relacionada con la estructura de nombres jerárquica de un objeto que se usa en la interface NML/EML.

El nombre de un objeto en MTNM es una lista de pares *<name,value>.* El campo *name* es un String dado por MTNM según el tipo de objeto y el campo *value* es un String con el nombre del objeto. A modo de ejemplo se lista el nombre de una Subnetwork Connection.

```
name="EMS";value="ACMEBank/ACMENet"
name="MultiLayerSubnetwork";value="sedeUruguay"
name="SubnetworkConnection";value="accesoNodoX"
```

La clase *NombreMtnm* implementa la función `toString()` que genera un String a partir de un `NameAndStringValue_T`. El formato del String fue especificado por nosotros y es el siguiente: *name* correspondiente al objeto seguido por SEPARADOR, concatenado con los sucesivos *value* que forman el nombre, separados por SEPARADOR. El ejemplo anterior quedaría (tomando como SEPARADOR el punto):

```
SubnetworkConnection.ACMEBank/ACMENet.sedeUruguay.accesoNodoX
```

La función `toString()` fue de gran utilidad en la implementación de todos los componentes y en particular en la interacción con el Inventario de red del proyecto Gesinv, el cual la utiliza como forma de generar un String que le permite identificar de forma única a un objeto.

Otro detalle importante de `NombreMtnm.java` es que se implementó como una clase *hashable* (implementa las operaciones `hashCode()` y `equals()`) lo que permite que objetos de esta clase puedan ser agregados a estructuras de hash dando la posibilidad de búsquedas eficientes.

Estructura de datos

Para brindar mayor flexibilidad se dejo pública la estructura ya que puede ser útil poderla cargar directamente.

```
public NameAndStringValue_T[] value;

public static String SEPARADOR=".";
```

La clase incluye tres constructores, el por defecto, un segundo constructor que recibe como parámetro un `NameAndStringValue_T[]` y un tercero que recibe un *String* con el nombre del objeto en el formato devuelto por la función `toString()` de la propia clase *NombreMtnm.* Este último realiza la operación inversa de dicha función, generando a partir de un *String* un `NameAndStringValue_T[]` y asignándolo a la estructura de datos *value* y tiene como pre-condición que el SEPARADOR no puede formar parte del *String* que se pasa como atributo ya que el mismo no es escapeado.

Para ver detalles relacionados a las operaciones implementadas referirse al Apéndice I: *Documentación de Capa de Elemento.*

8.4.2 MTNMDefs

Según la convención utilizada por MTNM cada tipo de objeto tiene un String que lo identifica y una estructura jerárquica de nombre, sin embargo estas especificaciones solo se dejan por escrito en la documentación de las IDLs pero no forman parte de ninguna interface o tipo. Por lo tanto para facilitar la implementación de los componentes se creo la clase `MTNMDefs.java`, que incluye los String de tipo y las funciones necesarias para construir el nombre de cada uno de los objetos MTNM.

Estructura de datos

Cada uno de los String de tipo según la convención MTNM:

```
public static String NAME_EMS="EMS";
public static String NAME_MULTILAYERSUBNETWORK="MultiLayerSubnetwork";
public static String NAME_SUBNETWORKCONNECTION="SubnetworkConnection";
public static String NAME_MANAGEDELEMENT="ManagedElement";
public static String NAME_TOPOLOGICALLINK="TopologicalLink";
public static String NAME_PTP="PTP";
public static String NAME_CTP="CTP";
public static String NAME_TPPOOL="TPPool";
public static String NAME_TRAFFICDESCRIPTOR="TrafficDescriptor";
public static String NAME_EQUIPMENTHOLDER="EquipmentHolder";
public static String NAME_EQUIPMENT="Equipment";
public static String NAME_PGP="PGP";
public static String NAME_AID="AID";
```

Para ver detalles relacionados a las operaciones implementadas referirse al Apéndice I: *Documentación de Capa de Elemento.*

8.5 EMS Manager

El Ems Manager es el componente de capa de elemento responsable de gestionar las Subnetwork (MLSN), los Topological Links (TL) y las alarmas de un sistema EMS. Dicho componente es una fachada de granularidad gruesa que permite el acceso a los objetos MTNM mencionados asi como también a los atributos del propio EMS.

Según el conjunto de IDLs MTNM el componente EMSMgr ofrece la interface EMSMgr_I que comprende las operaciones necesarias para recuperar los objetos anteriormente mencionados. Sin embargo como MTNM es una interface EML/NML no ofrece operaciones para crear, destruir o modificar Subnetwork y Topological Links, objetos que son responsabilidad de capa de elemento. Debido a esto se incorpora una nueva interface EMSMgrInternal_I especificada en la IDL emsMgrInternal.idl, la misma extiende la interface EMSMgr_I agregando las operaciones mencionadas anteriormente. A su vez la interface EMSMgr_I extiende la interface Common_I de la cual heredan las fachadas de capa de elemento y que ofrece un conjunto de operaciones comunes a todas.

Para mantener la consistencia de sus objetos el EmsMgr utiliza la interface ManagedElementMgr_I de otro componente de capa de elemento, el ManagedElementMgr.

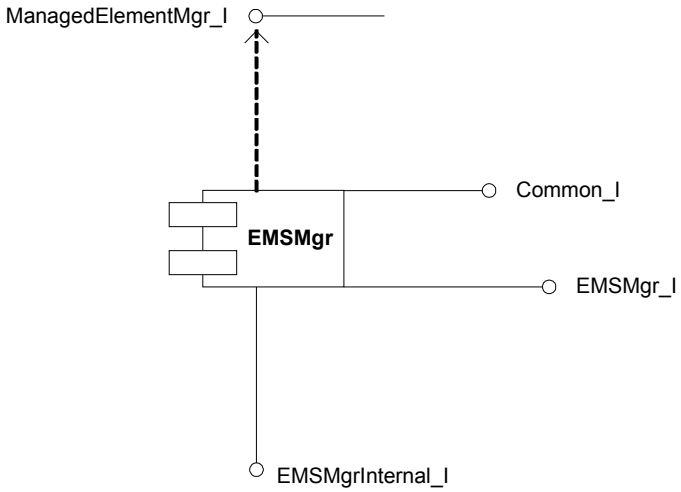


Diagrama 27 Contexto del componente EMSMgr

8.5.1 Alcance

Para cada una de las interfaces del componente EMSMgr se detallan sus operaciones indicando si las mismas fueron o no implementadas:

- EMSMgr_I

  Operaciones implementadas:

  - getAllTopLevelSubnetworkNames
  - getAllTopLevelSubnetworks
  - getAllTopLevelTopologicalLinkNames
  - getAllTopLevelTopologicalLinks
  - getEMS
  - getTopLevelTopologicalLink

Operaciones no implementadas:

- getAllEMSAndMEActiveAlarms
- getAllEMSSystemActiveAlarms

- Common_I

  Operaciones implementadas:

  - setNativeEMSName
  - setOwner
  - setUserLabel
  - getCapabilities

A continuación se detallan las operaciones incorporadas por la interface EMSMgrInternal_I.

- EMSMgrInternal_I
  - asignarEMS
  - crearTopLevelSubnetwork
  - crearTopLevelTopologicalLink
  - destruirTopLevelSubnetwork
  - destruirTopLevelTopologicalLink
  - modificarTopLevelSubnetwork
  - modificarTopLevelTopologicalLink

8.5.2 Inicialización

Al instanciar un componente EMSMgr, el mismo recibe en su constructor una lista de parámetros necesarios para su correcta inicialización. Los parámetros son los siguientes:

Parámetro	***Tipo***	***Valor por defecto***	***Descripción***
java_class	String		Es una etiqueta que indica al Loader la identidad del componente a levantar (en este caso, el EMSMgr)
ns_name	String		El nombre con el que se quiere que se registre el componente en el servicio de nombres de CORBA.
intervaloPing	Integer	8000	Cada cuantos milisegundos se se controlan los iteradores que el componente crea.
maxIteradores	Integer	1000	Cantidad máxima de iteradores por tipo de iterador.
iteradoresTimeout	Integer	28800000	Tiempo en milisegundos después del cual se elimina un iterador que no esta siendo utilizado..

Parámetro	*Tipo*	*Valor por defecto*	*Descripción*
archivoInit	String		Nombre del archivo XML desde donde el componente obtiene la definición de los objetos que gestiona (MLSNs y TLs) y la configuración de sus propios atributos (owner, user label, etc). En el prototipo el nombre del archivo es emsInit.xml , el mismo puede ser cambiado utilizando el archivo de configuración de capa de elemento.[32]
ManagedElement	String		Nombre de la interface ManagedElementMgr_I de la fachada ManagedElementMgr en el servicio de nombres de CORBA. Esta referencia es usada para mantener la consistencia de los objetos[33].

Tabla 5 Parámetros del EMSMgr

El constructor activa el control de iteradores y posteriormente obtiene la definición de los objetos a partir del archivo XML correspondiente.

8.5.3 Persistencia

La definición de los objetos gestionados por el componente EMSMgr es almacenada en el archivo XML utilizado durante la inicialización. El esquema del mismo fue definido en el archivo emsInitSchema.xsd , dicho esquema realiza una correspondencia entre la estructura de los objetos MTNM y una estructura XML. El formato del esquema es el siguiente:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="ems" type="EmsType"/>
<xsd:complexType name="EmsType">
  <xsd:sequence>
    <xsd:element name="additionalInfo" type="AdditionalInfoType" minOccurs="0"
maxOccurs="1"/>
    <xsd:element name="subNetworks" type="SubNetworksType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="topologicalLinks" type="TopologicalLinksType" minOccurs="0"
maxOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/> <!-- "CompanyName/EMSname" -->
  <xsd:attribute name="userLabel" type="xsd:string"/>
  <xsd:attribute name="nativeEMSName" type="xsd:string"/>
  <xsd:attribute name="owner" type="xsd:string"/>
  <xsd:attribute name="emsVersion" type="xsd:string"/>
  <xsd:attribute name="type" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="SubNetworksType">
  <xsd:sequence>
    <xsd:element name="subNetwork" type="SubNetworkType" minOccurs="0"
maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="TopologicalLinksType">
  <xsd:sequence>
    <xsd:element name="topologicalLink" type="TopologicalLinkType" minOccurs="0"
                 maxOccurs="unbounded"/>
```

32 Ver sección 8.1 Inicialización y configuración.

33 Ver sección 8.5.5 Detalles de implementación.

```
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="SubNetworkType">
  <xsd:sequence>
    <xsd:element name="supportedRates" type="layerRatesType" minOccurs="0"
maxOccurs="1"/>
    <xsd:element name="additionalInfo" type="AdditionalInfoType" minOccurs="0"
maxOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
  <xsd:attribute name="userLabel" type="xsd:string"/>
  <xsd:attribute name="nativeEMSName" type="xsd:string"/>
  <xsd:attribute name="owner" type="xsd:string"/>
  <xsd:attribute name="subnetworkType" type="xsd:int"/> <!-- Topology_T -->
</xsd:complexType>

<xsd:complexType name="TopologicalLinkType">
  <xsd:sequence>
    <xsd:element name="additionalInfo" type="AdditionalInfoType" minOccurs="0"
maxOccurs="1"/>
    <xsd:element name="aEndTPs" type="EndTPsType" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="zEndTPs" type="EndTPsType" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
  <xsd:attribute name="userLabel" type="xsd:string"/>
  <xsd:attribute name="nativeEMSName" type="xsd:string"/>
  <xsd:attribute name="owner" type="xsd:string"/>
  <xsd:attribute name="direction" type="xsd:int"/> <!-- ConnectionDirection_T -->
  <xsd:attribute name="rate" type="xsd:short"/>
</xsd:complexType>

<xsd:complexType name="layerRatesType">
  <xsd:sequence>
    <xsd:element name="layerRate" type="xsd:short" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="EndTPsType">
  <xsd:choice>
    <xsd:element name="PTP" type="PTPType" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="CTP" type="CTPType" minOccurs="1" maxOccurs="1"/>
  </xsd:choice>
</xsd:complexType>
<xsd:complexType name="PTPType">
  <xsd:attribute name="managedElement" type="xsd:string"/>
  <xsd:attribute name="ptp" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="CTPType">
  <xsd:attribute name="managedElement" type="xsd:string"/>
  <xsd:attribute name="ptp" type="xsd:string"/>
  <xsd:attribute name="ctp" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="AdditionalInfoType">
  <xsd:sequence>
    <xsd:element name="NameAndStringValue" type="NameAndStringValueType" minOccurs="0"
                 maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="NameAndStringValueType">
  <xsd:attribute name="name" type="xsd:string"/>
  <xsd:attribute name="value" type="xsd:string"/>
</xsd:complexType>
</xsd:schema>
```

Para obtener información acerca de XML/XSD ver Apéndice II: *XML/XSD*.

El diseño del esquema tiene como partida los tipos de MTNM EMS_T, MultiLayerSubnetwork_T y TopologicalLink_T definidos en el conjunto de IDLs MTNM.

8.5.4 Interface Interna

Para cubrir las operaciones no incluidas en MTNM se especificó una nueva interface en la IDL emsMgrInternal.idl. La misma hereda de EMSMgr_I y utiliza parte del conjunto de IDLs MTNM.

```
#ifndef emsMgrInternal_idl
#define emsMgrInternal_idl
// ******************************
// * emsMgrInternal.idl          *
// ******************************
//Include list
#include "globaldefs.idl"
#include "common.idl"
#include "emsMgr.idl"
#include "multiLayerSubnetwork.idl"
#include "notifications.idl"
#include "topologicalLink.idl"
#include "transmissionParameters.idl"
#include "subnetworkConnection.idl"
#include "managedElement.idl"
#include "topologicalLink.idl"
#include "terminationPoint.idl"
module mitiNum {
module ems{
module internal{
module emsMgrInternal
{
   interface  EMSMgrInternal_I : emsMgr::EMSMgr_I
   {
       void asignarEMS(in emsMgr::EMS_T emsInfo)
             raises(globaldefs::ProcessingFailureException);
       void crearTopLevelSubnetwork(
             in multiLayerSubnetwork::MultiLayerSubnetwork_T mlsn)
             raises(globaldefs::ProcessingFailureException);
       void modificarTopLevelSubnetwork(
             in multiLayerSubnetwork::MultiLayerSubnetwork_T mlsn)
             raises(globaldefs::ProcessingFailureException);
       void destruirTopLevelSubnetwork(
             in globaldefs::NamingAttributes_T mlsnName)
             raises(globaldefs::ProcessingFailureException);
       void crearTopLevelTopologicalLink(
             in topologicalLink::TopologicalLink_T tp)
             raises(globaldefs::ProcessingFailureException);
       void modificarTopLevelTopologicalLink(
             in topologicalLink::TopologicalLink_T tp)
             raises(globaldefs::ProcessingFailureException);
       void destruirTopLevelTopologicalLink(
             in globaldefs::NamingAttributes_T tpName)
             raises(globaldefs::ProcessingFailureException);
    };
};
};
};
};
#endif
```

La necesidad de incluir una nueva interface se plantea en una reunión entre ambos grupos de proyecto en la discusión de como inicializar y mantener la consistencia entre los componentes de capa de elemento EMSMgr y ManagedElementMgr, los que gestionan objetos que tienen estrecha relación.

Para iniciar el sistema los pasos a seguir son: crear Subnetworks en el EMSMgr, crear ManagedElements en el ManagedElementMgr y finalmente crear TopologicalLinks en el EMSMgr. Cada paso implica una comunicación entre ambos componentes de forma de crear sus objetos consistentemente.

Al crear una Subnetwork se le asocia una lista (posiblemente vacía) de LayerRate soportadas, luego al crear un ManagedElement se indica a que Subnetwork pertenece y que conjunto de Rate soporta y por ultimo al crear un TopologicalLink se le asocia un rate y un par de TPs cada uno con una lista de Layer/Rate soportadas, cada TP pertenece a un ManagedElement.

Cada paso tiene una dependencia con el anterior, al crear un ManagedElement debe existir la Subnetwork que lo contiene y por lo menos uno de sus Rate debe estar incluido en la lista de LayerRate de la misma. Un ManagedElement puede pertenecer a mas de una Subnetwork siempre que este en distintos layer rate, sin embargo dos Subnetwork no pueden solaparse en un mismo layer rate. Estas verificaciones son realizadas por el ManagedElementMgr y para ello obtiene del EMSMgr la lista de Subnetwork a través de la operación `EMSMgr_I.getAllTopLevelSubnetworks()`.

Al crear un TopologicalLink se debe verificar que su atributo Rate este incluido en las respectivas listas de Layer/Rate de los TPs cuyos nombres se incluyen en los atributos aEndTP y zEndTP. Para ello el EMSMgr obtiene ambos TPs del ManagedElementMgr invocando `ManagedElementMgr_I.getTP()` una vez por cada uno, pasando como parámetro sus nombres.

Los objetos creados son aquellos que cumplen las dependencias mencionadas anteriormente y en el caso de Subnetworks y TopologicalLinks pasan a formar parte del archivo XML desde donde el componente EMSMgr levanta su configuración y mantiene su persistencia.

Para configurar los objetos del EMSMgr se creo una aplicación gráfica con el objetivo de crear, modificar y eliminar Subnetworks y TopologicalLinks, la misma se llama emsConfigGUI. La emsConfigGUI establece una sesión con capa de elemento y utiliza la interface del EMSMgr para realizar las tareas mencionadas. Referirse al Apéndice VII: *Manual de usuario EMS CONFIG GUI*.

8.5.5 Detalles de implementación

La implementación de las interfaces incluye las siguientes generalidades:

- La gestión de los iteradores del componente se realiza de la misma forma que para todo el sistema.[34]
- Cada vez que un objeto, Subnetwork o TopologicalLink es creado, modificado o destruido se dispara una notificación empleando la clase `NotifProveedor`, enviando un evento de clase `NotifStructEvent`.
- El componente tiene una estructura de datos interna que carga a partir del archivo XML donde mantiene su persistencia.
- Las operaciones de consulta especificadas en la interface EMSMgr_I de MTNM leen de la estructura de datos interna y no del archivo XML. Esto permite el remplazo del XML por alguna otra tecnología, por ejemplo una base de datos sin tener que modificar la implementación de las mismas.
- Las operaciones de alta, baja y modificaciones ligadas a la persistencia trabajan directamente sobre el archivo XML a través del paquete de clases generado al compilar su esquema. Antes de finalizar invocan un procedimiento privado que actualiza la estructura de datos interna a partir del XML. En caso de sustituir el archivo por otra forma de persistencia las mismas se deben volver a implementar.

34 Ver sección 8.3 Iteradores.

Estructura de datos

La estructura de datos del componente se seleccionó de manera de facilitar la recuperación de los objetos del mismo, asi como también el acceso a sus atributos. La estructura es la siguiente:

```
private EMS_T _ems=null;
private List _multiLayerSubnetwork=new LinkedList();
private Map _topologicalLinks=new HashMap();
```

Para almacenar los atributos del EMS se utiliza un objeto de clase EMS_T especificada en el conjunto de IDLs MTNM y que es un tipo estructurado que agrupa todos los atributos del EMS (owner, user label, etc).

Se utilizó una lista para almacenar las Subnetworks ya que ambas operaciones de consulta, `getAllTopLevelSubnetworkNames()` y `getAllTopLevelSubnetworks()` devuelven listas de Subnetwork y no un objeto Subnetwork en particular.

En caso de los TopologicalLinks se consideró más adecuado utilizar un Hash con clave nombre de TopologicalLink para facilitar la implementación de la operación `getTopLevelTopologicalLink()` que recibe como parámetro (entre otros) el nombre del TopologicalLink a recuperar. Dicha estructura igual permite implementar de forma eficiente las operaciones masivas `getAllTopLevelTopologicalLinkNames()` y `getAllTopLevelTopologicalLinks()` ya que se pide al Hash un iterador (de java) para recorrerlo de forma secuencial.

Para ver detalles relacionados a las operaciones públicas y privadas implementadas referirse al Apéndice I: *Documentación de Capa de Elemento.*

8.6 MultiLayerSubnetwork Manager

El MultiLayerSubnetworkMgr es el componente de capa de elemento responsable de gestionar las SubnetworkConnections, y de mantener las relaciones existentes entre MultiLayerSubnetworks, SubnetworkConnections, TopologicalLinks y ManagedElements, según MTNM.

Dicho componente es una fachada de granularidad gruesa que permite el acceso a los objetos MTNM mencionados, entre otros, así como también a sus atributos. Hay que hacer aquí la salvedad de que no todos lo atributos de los objetos son seteables desde esta interface, sino solamente de las SubnetworkConnections, que son los objetos que el componente esta autorizado a crear.

Según el conjunto de IDLs MTNM el componente MultiLayerSubnetworkMgr ofrece la interface MultiLayerSubnetworkMgr_I que comprende las operaciones necesarias para recuperar los objetos anteriormente mencionados, y además poder administrar SNCs. La interface extiende la Common_I de la cual heredan todas las fachadas de capa de elemento y que ofrece operaciones y atributos comunes a todas.

Para mantener la consistencia de los objetos, el MultiLayerSubnetworkMgr utiliza las interfaces EMSMgr_I y ManagedElementMgr_I de capa de elemento, que son implementadas por los componentes EmsMgr y ManagedElementMgr.

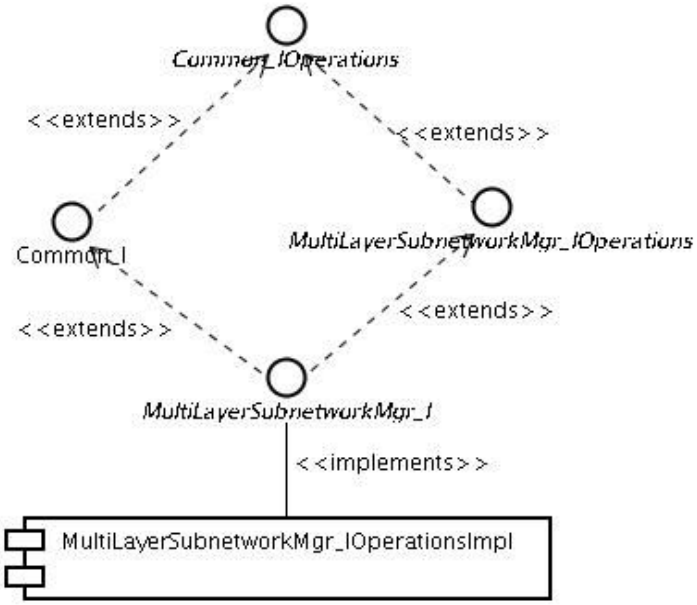


Diagrama 28 Contexto del componente MLSNMgr

8.6.1 Alcance

Para cada una de las interfaces del componente MultiLayerSubnetworkMgr se detallan sus operaciones indicando si las mismas fueron o no implementadas:

- Common_I

  Operaciones implementadas:

  - setNativeEMSName
  - setOwner
  - setUserLabel
  - getCapabilities

- MultiLayerSubnetwork_I

  Operaciones Implementadas:

  - activateSNC
  - createAndActivateSNC
  - checkValidSNC
  - createAndActivateSNC
  - createSNC
  - deactivateAndDeleteSNC
  - deactivateSNC
  - deleteSNC
  - getAllEdgePointNames
  - getAllEdgePoints
  - getAllManagedElementNames
  - getAllManagedElements
  - getAllManagedElementNames
  - getAllSubnetworkConnectionNames
  - getAllSubnetworkConnectionNamesWithTP
  - getAllSubnetworkConnections
  - getAllSubnetworkConnectionNamesWithTP
  - getAllSubnetworkConnections
  - getAllSubnetworkConnectionsWithTP
  - getAllTopologicalLinkNames
  - getAllTopologicalLinks
  - getMultiLayerSubnetwork
  - getRoute
  - getSNC
  - getSNCsByUserLabel
  - getTopologicalLink

  Operaciones no Implementadas:

  - getAllTPPoolNames
  - getAllTPPools
  - getAssociatedTP
  - getTPGroupingRelationships

8.6.2 Inicialización

Al instanciar un componente MultiLayerSubnetworkMgr, el mismo recibe en su constructor una lista de parámetros necesarios para su correcta inicialización. Los parámetros son los siguientes:

Parámetro	*Tipo*	*Valor por defecto*	*Descripción*
java_class	String		Es una etiqueta que indica al Loader la identidad del componente a levantar (en este caso, el MultiLayerSubnetworkMgr)
ns_name	String		El nombre con el que se quiere que se registre el componente en el servicio de nombres de CORBA.
intervaloPing	Integer	8000	Cada cuantos milisegundos se se controlan los iteradores que el componente crea.
maxIteradores	Integer	1000	Cantidad máxima de iteradores por tipo de iterador.
iteradoresTimeout	Integer	28800000	Tiempo en milisegundos después del cual se elimina un iterador que no esta siendo utilizado..
archivoInit	String		Nombre del archivo XML desde donde el componente obtiene las SubnetworkConnections que gestiona. En el prototipo el nombre del archivo es sncInit.xml[35], el mismo puede ser cambiado utilizando el archivo de configuración de capa de elemento.[36]
EMS	String	EMS.proyGrado	Nombre de la interface de la fachada EMSMgr en el servicio de nombres de CORBA.
ManagedElement	String	ManagedElement.proyGrado	Nombre de la interface ManagedElementMgr_I de la fachada ManagedElementMgr en el servicio de nombres de CORBA. Parte del proyecto Gesinv.
InternalMEM	String	InternalMEM.Servidor	Nombre de la interface interna en el servicio de nombres de CORBA. Esta interface permite manipular los objetos CrossConnect y es parte del proyecto Gesinv.

Tabla 6 Parámetros del MultiLayerSubnetworkMgr

El constructor activa el control de iteradores y posteriormente obtiene la definición de los objetos a partir del archivo XML correspondiente. Además el componente obtiene los objetos restantes, mediante interacciones con otras fachadas de capa de elemento.

35 Ver sección 8.6.3 Persistencia.

36 Ver sección 8.1 Inicialización y configuración.

A modo de descripción del trabajo hecho al levantar el componente, presentamos los pasos que realiza :

1. Obtiene las MultiLayerSubnetworks desde el EMSMgr_I
2. Obtiene los TopologicalLinks desde el EMSMgr_I
3. Obtiene los ManagedElements desde el ManagedElementMgr_I
4. Carga las SubnetworkConnections desde el archivo xml descripto anteriormente.
5. Con los datos recabados en los pasos 1 a 4 se carga una estructura auxiliar[37], con la cual el componente trabaja a lo largo de su vida en el sistema.

A continuación mostramos un diagrama de cada paso de la inicialización :

Paso 1 - Obtener las MultiLayerSubnetworks desde el EMSMgr_I

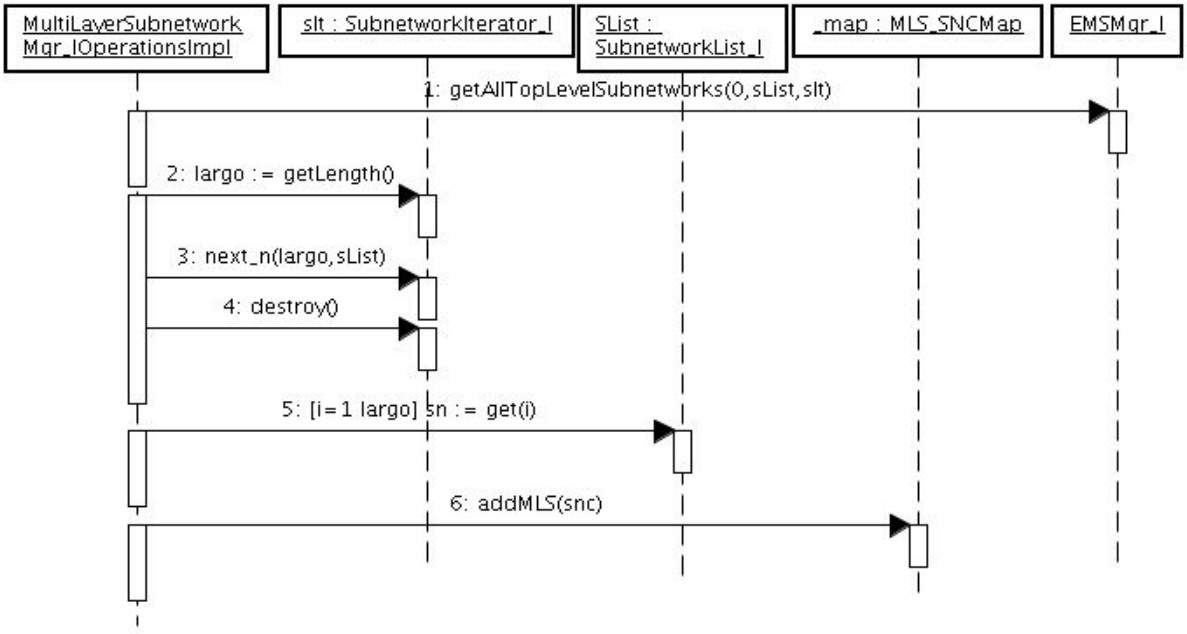


Diagrama 29 Inicialización - Paso 1

37 Ver sección 8.6.5 Estructura de datos.

Paso 2 - Obtener los TopologicalLinks desde el EMSMgr_I

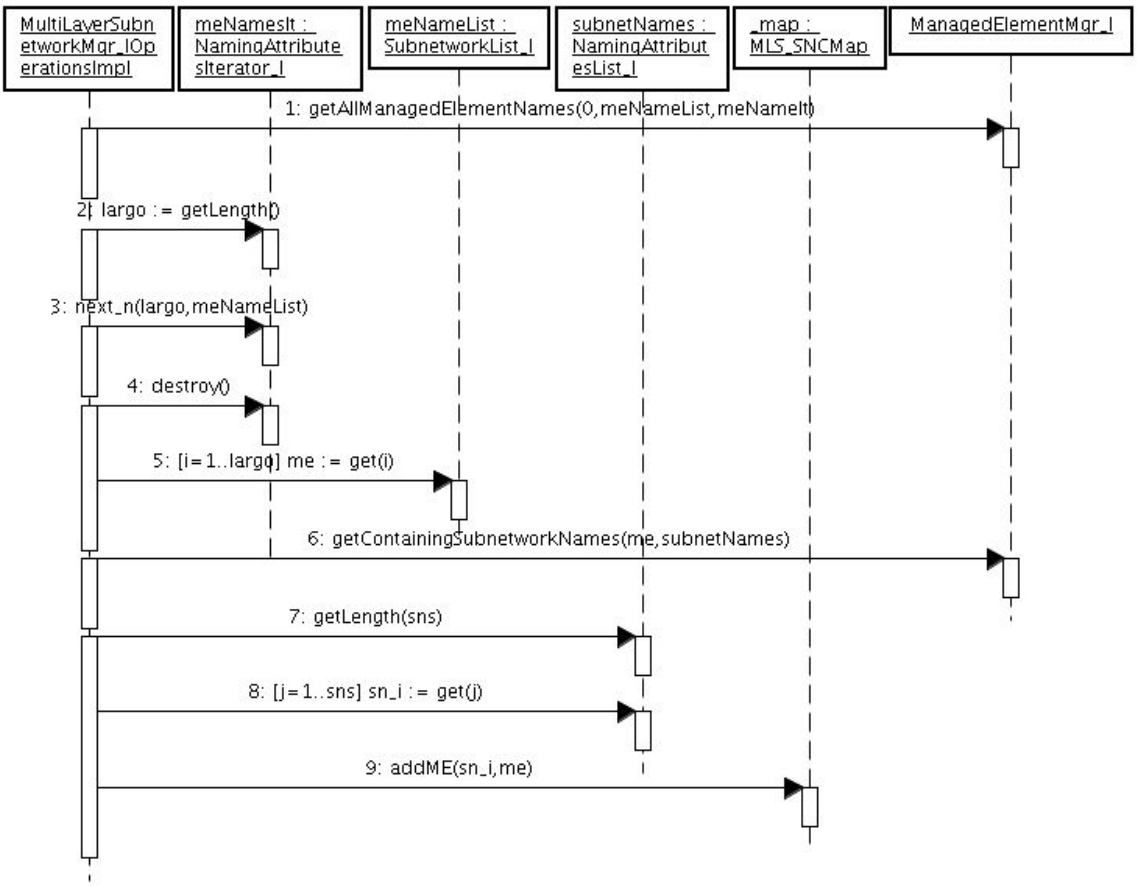


Diagrama 30 Inicialización - Paso 2

Paso 3 - Obtener los ManagedElements desde el ManagedElementMgr_I

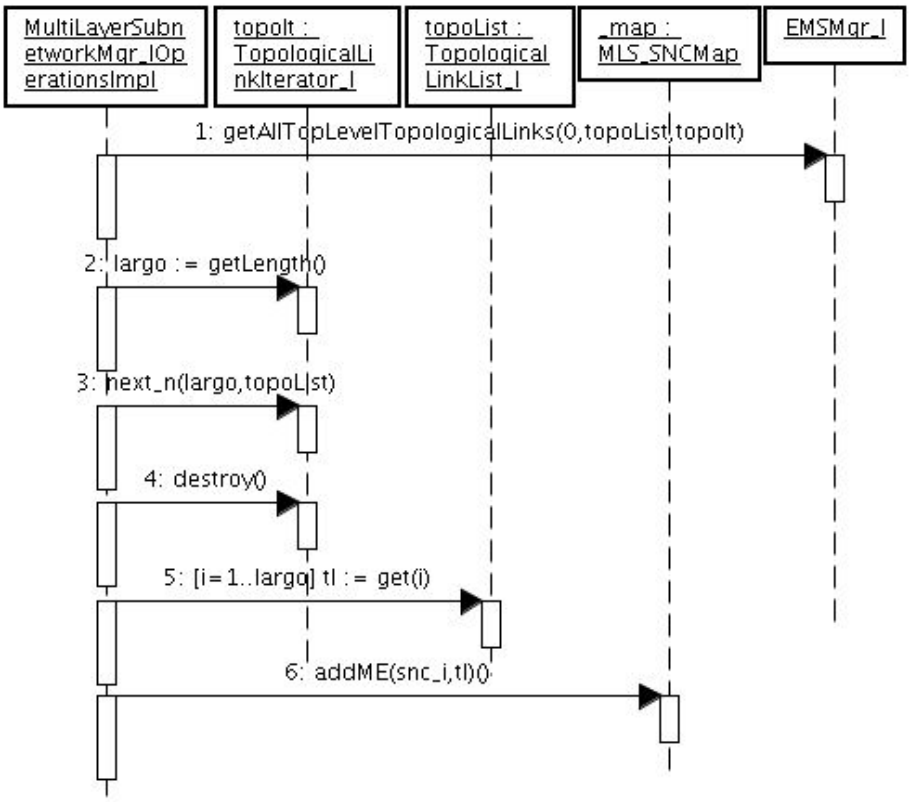


Diagrama 31 Inicialización - Paso 3

Paso 4 - Cargar las SubnetworkConnections desde un archivo

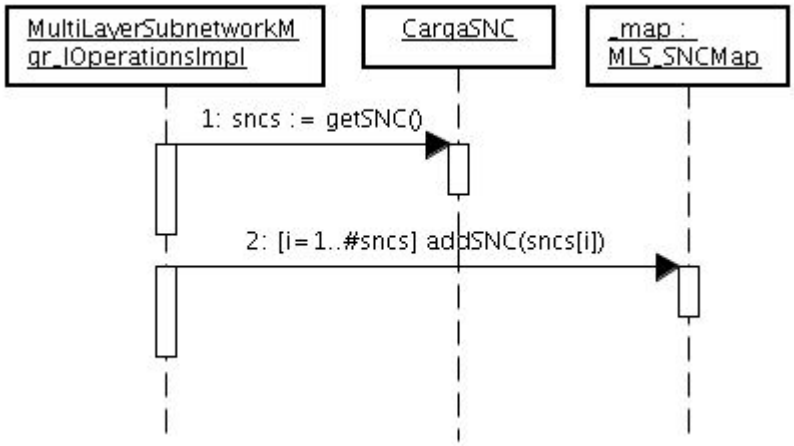


Diagrama 32 Inicialización - Paso 4

8.6.3 Persistencia

La persistencia esta dada por dos clases auxiliares, BackupSNC y CargarSNC, que se encargan de hacer persistentes y de cargar respectivamente, las SNCs que existen en el sistema. Como se ve, solo se encarga de respaldar la información de SubnetworkConnections, que es la información que la clase MultiLayerSubnetworkMgr_IoperationsImpl administra. Los demás objetos son persistidos por las interfaces responsables de ellos.

La persistencia esta implementada mediante la escritura de archivos xml. Estas escrituras se hace solamente cuando es relevante, por ejemplo cuando una SubnetworkConnection es creada, o el estado de alguna fue modificado.

La información almacenada por la persistencia es usada para la inicialización[38]. El esquema del archivo XML fue definido en el archivo sncInitSchema.xsd , dicho esquema realiza una correspondencia entre la estructura de los objetos MTNM y una estructura XML. El formato del esquema es el siguiente:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<!-- Define la coleccion de SNCs -->
<xsd:element name="subnetworkconnections" type="SNCColeccion"/>

<!-- Define la coleccion de subnetwork Connections -->
<xsd:complexType name="SNCColeccion">
    <xsd:sequence>
        <xsd:element name="subnetworkconnection" type="SNCType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>

<!-- Define la subnetwork Connection -->
<xsd:complexType name="SNCType">
    <xsd:sequence>
        <xsd:element name="sncmtnmname" type="NombreMTNMType" minOccurs="1"
maxOccurs="1"/>
        <xsd:element name="aditinfo" type="NombreMTNMType" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="aendtp" type="TPDataType" minOccurs="1"
maxOccurs="unbounded"/>
        <xsd:element name="zendtp" type="TPDataType" minOccurs="1"
maxOccurs="unbounded"/>
        <xsd:element name="crossconnect" type="CrossConnectType" minOccurs="1"
maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="owner" type="xsd:string"/>
    <xsd:attribute name="userlabel" type="xsd:string"/>
    <xsd:attribute name="nativeemsname" type="xsd:string"/>
    <xsd:attribute name="sncstate" type="xsd:string"/>
    <xsd:attribute name="direction" type="xsd:string"/>
    <xsd:attribute name="rate" type="xsd:string"/>
    <xsd:attribute name="staticprotectionlevel" type="xsd:string"/>
    <xsd:attribute name="snctype" type="xsd:string"/>
    <xsd:attribute name="reroutedallowed" type="xsd:string"/>
    <xsd:attribute name="networkrouted" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="TPDataType">
    <xsd:sequence>
       <xsd:element name="nomendtp" type="NombreMTNMType" minOccurs="1" maxOccurs="1"/>
       <xsd:element name="ingrtraffdescname" type="NombreMTNMType" minOccurs="0"
maxOccurs="1"/>
```

38 Ver sección 8.6.2 Inicialización.

```
       <xsd:element name="egrtraffdescname" type="NombreMTNMType" minOccurs="0"
maxOccurs="1"/>
       <xsd:element name="transmparam" type="LayeredParamType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="tpmappingmode" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="CrossConnectType">
    <xsd:sequence>
       <xsd:element name="additinfo" type="NombreMTNMType" minOccurs="1" maxOccurs="1"/>
       <xsd:element name="aendtp" type="NombreMTNMType" minOccurs="1"
maxOccurs="unbounded"/>
       <xsd:element name="zendtp" type="NombreMTNMType" minOccurs="1"
maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="active" type="xsd:boolean"/>
    <xsd:attribute name="direction" type="xsd:string"/>
    <xsd:attribute name="cctype" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="LayeredParamType">
    <xsd:sequence>
       <xsd:element name="transmparamname" type="NombreMTNMType" minOccurs="1"
maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="layer" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="NombreMTNMType">
    <xsd:sequence>
       <xsd:element name="nom" type="NombreValor" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="NombreValor">
    <xsd:attribute name="nombre" type="xsd:string"/>
    <xsd:attribute name="valor" type="xsd:string"/>
</xsd:complexType>


</xsd:schema>
```

Para obtener información acerca de XML/XSD ver Apéndice II: *XML/XSD*.

El diseño del esquema tiene como partida los tipos de MTNM SubnetworkConnection_T, NameAndStringValue_T y otros definidos en el conjunto de IDLs MTNM.

8.6.4 Detalles de implementación

La implementación de las interfaces incluye las siguientes generalidades:

- La gestión de los iteradores del componente se realiza de la misma forma que para todo el sistema.[39]
- Cada vez que un objeto, SubnetworkConnection_T es creado, modificado o destruido se dispara una notificación empleando la clase `NotifProveedor`, enviando un evento de clase `NotifStructEvent`.

39 Ver sección 8.3 Iteradores.

- El componente tiene una estructura de datos interna que le permite responder a los requerimientos especificados en la interface MultiLayerSubnetworkMgr_I de MTNM de manera eficiente. Esta estructura es cargada en la inicialización del componente como se explica en la sección correspondiente.
- Las operaciones de alta, baja y modificaciones ligadas a las SNCs disparan la persistencia de la estructura auxiliar mencionada en el párrafo anterior, escribiendo directamente sobre el archivo XML. Antes de finalizar, invocan un procedimiento privado que actualiza el archivo XML. En caso de sustituir el archivo por otra forma de persistencia las mismas se deben volver a implementar.

8.6.5 Estructura de datos

La estructura auxiliar mencionada anteriormente en varias oportunidades, fue diseñada de manera de facilitar la recuperación de los objetos del mismo, el acceso a sus atributos y representar las restricciones impuestas por el modelo MTNM.

Esta clase auxiliar `MLS_SNCMap` se vale a su vez de algunas otras clases, que representan los objetos relevantes, como son los MEs y los TLs.

La estructura consta de dos hashes indexados por el nombre MTNM de cada objeto. En uno de ellos la clave es el nombre de una MultiLayerSubnetwork, y en el otro el nombre de una SubnetworkConnection.

Estas decisiones radican en el hecho de que las búsquedas de objetos dado su nombre son frecuentes, y dado que el nombre MTNM de un objeto es único, este es usado como clave.

Con esto logramos buscar, agregar o eliminar en orden 1 promedio.

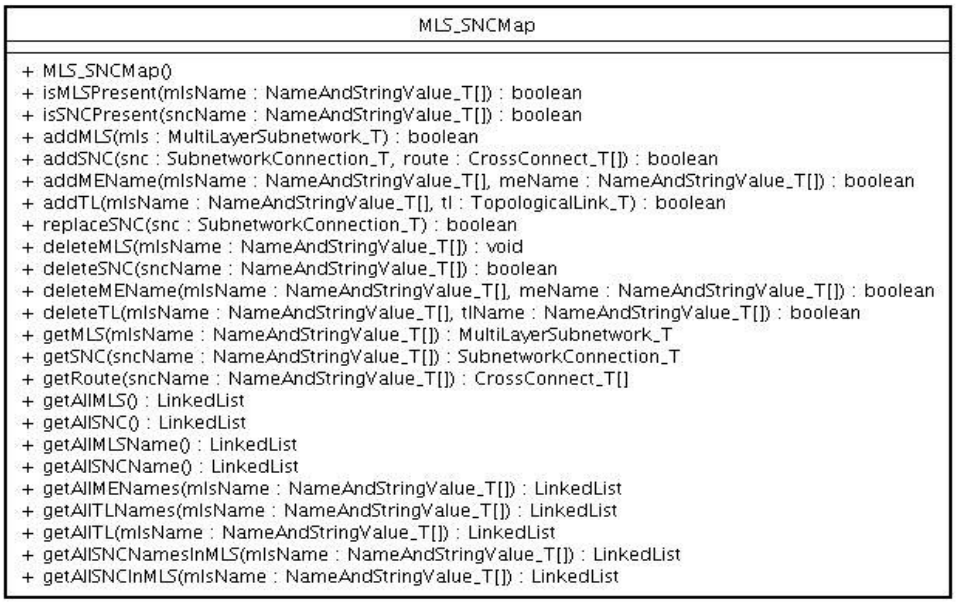


Diagrama 33 Clase auxiliar MLS_SNCMap

Para ver detalles relacionados a las operaciones públicas y privadas implementadas referirse al Apéndice I: *Documentación de Capa de Elemento.*

8.6.6 Interacción con Gesinv

La interacción con el grupo Gesinv se da a traves de dos caminos, vía MTNM y vía una interface interna definida en común que es la InternalMEM.

Ambas son interfaces CORBA, y son la forma que tiene la clase MultiLayerSubnetworkMgr_IOperationsImpl de obtener información acerca de los MEs, y de eventualmente modificar o crear objetos.

8.6.7 Asunciones

- Se asume que las MultiLayerSubnetworks que maneja la clase son las mismas que maneja el EMSMgr. Esto no es necesariamente cierto en lo que respecta a MTNM, sino que la relación entre las MultiLayerSubnetworks es de inclusión, de manera que las que tiene el EMSMgr están incluidas en las que tiene el MultiLayerSubnetworkMgr_IOperationsImpl.

  En nuestra realidad, a propósito de un prototipo, se asume lo anterior, con base en que tendremos un numero bastante acotado de MultiLayerSubnetworks.

  Esto repercute básicamente en la creación de SubnetworkConnections, simplificando la creación de las rutas a nivel de capa de red(NMS).

  Si se quisiera cumplir estrictamente con MTNM se debería :

  - Persistir las MultiLayerSubnetworks en la clase.
  - Al levantar la clase, obtener las MultiLayerSubnetworks desde la persistencia.
  - Preguntar al EMSMgr sobre las MultiLayerSubnetworks que son TopLevel.
- Algo similar ocurre con los TopologicalLinks, que también se supone son los mismos que los manejados por el EMSMgr. Esto no es necesariamente cierto en lo que respecta a MTNM, sino que la relación entre los TopologicalLinks es de inclusión, de manera que los que tiene el EMSMgr están incluidos en las que tiene el MultiLayerSubnetworkMgr_IOperationsImpl.

  En nuestra realidad, a propósito de un prototipo, se asume lo anterior, casi como una consecuencia de la primera asunción.

  Esto repercute básicamente en la creación de SubnetworkConnections, simplificando la creación de las rutas a nivel de capa de red(NMS).

  Si se quisiera cumplir estrictamente con MTNM se debería :

  - Persistir los TopologicalLinks en la clase.
  - Al levantar la clase, obtener los TopologicalLinks desde la persistencia.
  - Preguntar al EMSMgr sobre los TopologicalLinks que son TopLevel.

Hay otras asunciones hechas que tienen mas que ver con los Managed Elements, pero que tienen importantes implicancias en el funcionamiento de esta fachada.

- CrossConnect

  mitiNum tiene algunas restricciones sobre estos objetos.

  - Shared

    No se permite que los CrossConnect sean compartidos por SubnetworkConnections.
- TPPools

  mitiNum no implementa este tipo particular de Tps. Los TPPool son de ayuda principalmente a nivel administrativo[40].
- EMSFreedomLevel

  Usado en la administración de las SubnetworkConnections, tiene importancia en el comportamiento del EMS frente a como proceder.

  El EMSFreedomLevel soportado por el sistema es EMSFL_CC_AT_SNC_LAYER[41].

40 Ver definición en el documento de MTNM TMF513V2_0.pdf 4.1.6

- GradesOfImpact[42]

  No se toma en cuenta este parámetro en las operaciones relacionadas a las SubnetworkConnections.

- Restricciones al crear SubnetworkConnections.

  Al crear SubnetworkConnections se debe aportar una estructura SNCCreateData_T con los datos requeridos por la fachada para crearla. Hay importantes restricciones hechas para los fines del proyecto.

  - ConnectionDirection[43]

    Tomamos las SubnetworkConnections como bidireccionales, usando CD_BI.

  - Staticprotectionlevel[44]

    Se soporta el nivel UNPROTECTED.

  - Reroute[45]

    Se soporta el RR_NA.

  - Networkrouted[46]

    Se soporta el NR_NA.

  - SNCType[47]

    Se soporta ST_SIMPLE.

  - Parámetro ccInclusions

    Suponemos que los Crossconnects que vienen aquí representan la ruta completa de la SubnetworkConnection a crear.

  - Parámetro neTpInclusions.

    No es tomado en cuenta.

  - Parámetro neTpSNCExclusions.

    No es tomado en cuenta.

  - Parámetros aEnd y zEnd.

    Aunque en teoría en cada uno de estos parámetros se permite el nombre de varios TPs, sólo se toma en cuenta el primero de ellos, de manera que las SubnetworkConnection creada es de un TP a otro.

41 Ver mas detalles sobre EMSFreedomLevel_T en el documento de MTNM TMF814v2_0.zip
42 Ver mas detalles sobre GradesOfImpact_T en el documento de MTNM TMF814v2_0.zip
43 Ver mas detalles sobre ConnectionDirection_T en el documento de MTNM TMF814v2_0.zip
44 Ver mas detalles sobre SNCTypes en el documento de MTNM TMF814v2_0.zip
45 Ver mas detalles sobre Reroute_T en el documento de MTNM TMF814v2_0.zip
46 Ver mas detalles sobre NetworkRouted en el documento de MTNM TMF814v2_0.zip
47 Ver mas detalles sobre SNCType en en el documento de MTNM TMF814v2_0.zip

8.7 Notificaciones

Las aplicaciones distribuidas requieren un modelo de comunicación fácilmente escalable que desacople proveedores de consumidores y simultáneamente soporte propiedades relacionadas con calidad de servicio (QoS) y mecanismos de filtrado de eventos. El Servicio de Notificaciones de CORBA proporciona un mecanismo publish/subscribe diseñado para soportar de forma escalable comunicación mediante eventos. El servicio incluye ruteo eficiente de eventos entre muchos proveedores y consumidores, ofreciendo varias propiedades de QoS y filtrado de eventos en puntos múltiples del sistema.

El Servicio de Notificaciones es una extensión (por herencia) del Servicio de Eventos de CORBA, este último ofrece una forma de comunicación desacoplada, asincrónica y transparente entre participantes. El Servicio de Eventos define tres roles:

- Proveedores: parte que produce eventos.
- Consumidores: parte que consume eventos.
- Canales de eventos: mediador a traves del cual múltiples consumidores y proveedores se comunican asincrónicamente.

Sin embargo el Servicio de Eventos tiene limitaciones: no ofrece interfaces o políticas para soportar propiedades de QoS y no tiene un mecanismo de filtrado de eventos poderoso, estas limitaciones fueron levantadas en el Servicio de Notificaciones.

La estructura de componentes del Servicio de Notificaciones de CORBA se ilustra en el Diagrama 34 y cada uno de los componentes se describe brevemente a continuación:

- Evento estructurado: define una estructura de datos estándar en la que se puede enviar una gran variedad de mensajes.
- Objetos proxys: Son objetos que ofrecen interfaces complementarias a los clientes. Por ejemplo un consumidor obtiene y se conecta a un proxy proveedor y un proveedor obtiene y se conecta a un proxy consumidor. Por lo tanto, un proveedor envía eventos a su proxy consumidor, mientras que un consumidor recibe eventos desde su proxy proveedor. Esta abstracción permite conectividad anónima entre consumidores y proveedores.
- Objetos admin: Un objeto admin es una factory que crea interfaces proxy a las que un cliente se conectará. Los admin consumidores crean proxy proveedores a los que se conectaran los consumidores y los admin proveedores crean proxy consumidores a los que se conectaran los proveedores. Si un canal soporta múltiples admin se pueden tener grupos de proxys según el tipo de información a la cual un consumidor desea suscribirse.
- Objetos filtros: Un filtro puede ser asociado con todos los objetos admin y proxy o solo a uno en particular. Estos regulan el envío y la recepción de eventos.
- Canal de eventos: Un canal de eventos es un factory que crea objetos admin consumidores y admin proveedores. Esta es una diferencia con el Servicio de Eventos que solo soporta un tipo de objetos admin.
- Propiedades de QoS: Son tuplas <*nombre, valor*> que contienen propiedades especificadas por los usuarios. Las propiedades de QoS pueden ser asociadas con un canal de eventos, un objeto admin, un objeto proxy y un mensaje de evento individual.

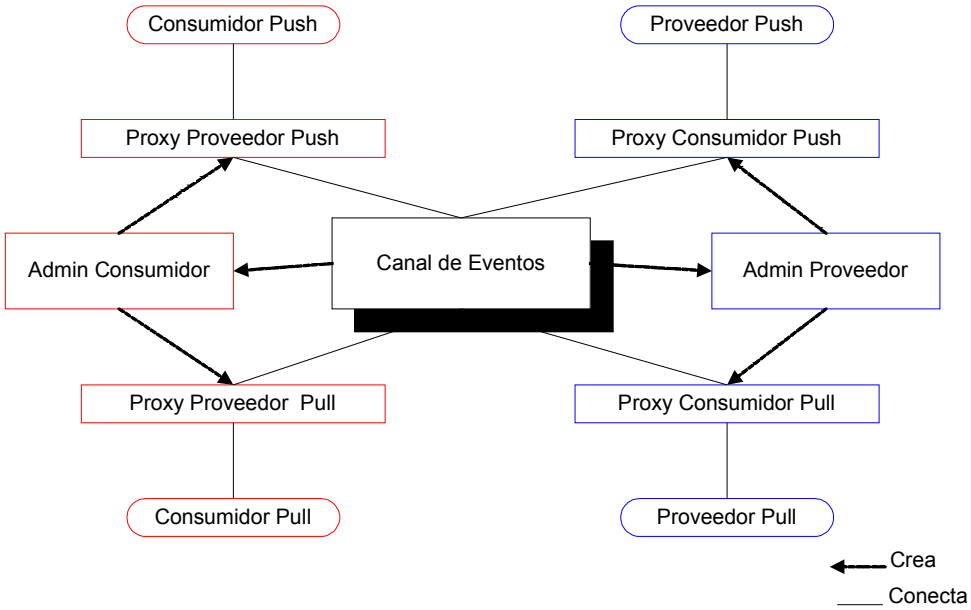


Diagrama 34 Componentes del Servicio de Notificaciones de CORBA

El Servicio de Notificaciones soporta dos modelos de envío de eventos: modelo push y modelo pull. En el modelo push el consumidor espera en forma pasiva la llegada de eventos. El proveedor genera activamente eventos y los envía al canal, el que posteriormente invoca al consumidor. En el modelo pull un consumidor activamente requiere un evento. Esta es una descripción simplificada de ambos modelos sin incluir los objetos proxy.

El Servicio de Notificaciones es la forma de comunicación utilizada por MTNM y las recomendaciones para su uso se especifícan en el documento TMF814v2.0, puntualmente en *notificationServiceUsage.html* incluido en la documentación de soporte[48]. Este documento incluye recomendaciones generales y especifíca detalladamente como se debe completar cada una de los campos de un evento estructurado. La siguiente es parte de la lista de recomendaciones y solo incluye las que se tuvieron en cuenta en nuestro proyecto:

- Usar solamente el modelo Push.
- Utilizar un solo canal por EMS para evitar problemas de orden en la llegada de eventos.
- Usar solo un Servicio de Notificaciones por EMS.

48 Ver documentación de MTNM en el CD de instalación de mitiNum.

Para finalizar en el Diagrma 35 se ilustra la estructura de un evento estructurado genérico como paso previo a la especificación de un evento en el contexto de la interface MTNM[49].

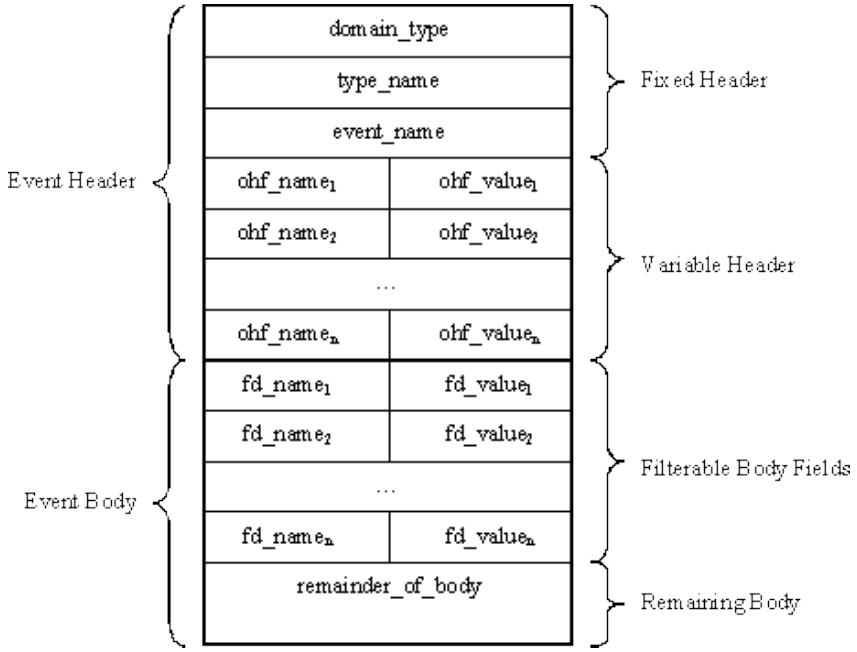


Diagrama 35 Estructura de un evento estructurado

8.7.1 Alcance

La especificación de un evento en el contexto de la interface MTNM cubierta en el documento TMF814v2.0, mas precisamente en *notificationServiceUsage.html* se tomo como base. Según la misma existen varios tipos de eventos estructurados que se especifican en el campo *type_name* del cabezal del evento. En la Tabla 7 se presentan los tipos de eventos definidos por MTNM, la columna alcance indica si el tipo correspondiente fue implementado en nuestro proyecto. Posteriormente para los tipos implementados se describe el contenido del cuerpo del evento (filterable_data y remainder_of_body) según la especificación MTNM indicando las modificaciones o adaptaciones realizadas para nuestro proyecto.

Evento *(type_name)*	**Alcance**
NT_OBJECT_CREATION	X
NT_OBJECT_DELETION	X
NT_ATTRIBUTE_VALUE_CHANGE	X
NT_STATE_CHANGE	X
NT_ROUTE_CHANGE	
NT_PROTECTION_SWITCH	
NT_TCA _ALERT	
NT_ALARM	
NT_FILE_TRANSFER_STATUS	

Tabla 7 Tipos de eventos especificados por MTNM

49 Ver sección 8.7.1 Alcance.

- Para *type_name* NT_OBJECT_CREATION la especificación MTNM se siguió sin modificaciones. Sin embargo los campos en que se prestó especial atención por su utilidad fueron *objectName, objectType* y el *remainder_of_body* donde se envía el objeto creado.

filterable_data		
Nombre	**Tipo de dato**	**Descripción**
notificationId	string	No están garantizadas la unicidad y la secuencia del notificationid.
objectName	globaldefs::NamingAttributes_T	Nombre del objeto que fue creado.
objectType	notifications::ObjectType_T	Tipo del objeto que fue creado.
emsTime	globaldefs::Time_T	Tiempo en el que el evento fue reportado en el sistema EMS.
neTime	globaldefs::Time_T	Tiempo reportado por el NE. En caso de no ser reportado el campo debe ir vacío.
edgePointRelated	boolean	TRUE en caso de ser un evento relacionado a un PTP que es edge point, FALSE en cualquier otro caso.
remainder_of_body		
El objeto creado según el tipo especificado en *objectType.* El objeto viaja dentro de un tipo Any de CORBA.		

- Para *type_name* NT_OBJECT_DELETION la especificación MTNM se siguió sin modificaciones. Sin embargo los campos en que se prestó especial atención por su utilidad fueron *objectName* y *objectType.*

filterable_data		
Nombre	**Tipo de dato**	**Descripción**
notificationId	string	No están garantizadas la unicidad y la secuencia del notificationid.
objectName	globaldefs::NamingAttributes_T	Nombre del objeto que fue borrado.
objectType	notifications::ObjectType_T	Tipo del objeto que fue borrado.
emsTime	globaldefs::Time_T	Tiempo en el que el evento fue reportado en el sistema EMS.
neTime	globaldefs::Time_T	Tiempo reportado por el NE. En caso de no ser reportado el campo debe ir vacío.
edgePointRelated	boolean	TRUE en caso de ser un evento relacionado a un PTP que es edge point, FALSE en cualquier otro caso.
remainder_of_body		
NULL		

- Para *type_name* NT_ATTRIBUTE_VALUE_CHANGE la especificación MTNM sufrió cambios. El *remainder_of_body* fue modificado respecto a la especificación MTNM. Según MTNM debía ir en NULL sin embargo en el marco de nuestro proyecto se empleo para enviar el objeto modificado, de esta forma se simplificó la implementación enviando los atributos modificados en el propio objeto. Por lo anterior el campo *attributeList* queda obsoleto. Para una justificación mas detallada acerca de dicha decisión referirse a la sección 1.2 *Detalles de implementación*.

filterable_data		
Nombre	**Tipo de dato**	**Descripción**
notificationId	string	No están garantizadas la unicidad y la secuencia del notificationid.
objectName	globaldefs::NamingAttributes_T	Nombre del objeto cuyos atributos fueron modificados.
objectType	notifications::ObjectType_T	Tipo del objeto cuyos atributos fueron modificados.
emsTime	globaldefs::Time_T	Tiempo en el que el evento fue reportado en el sistema EMS.
neTime	globaldefs::Time_T	Tiempo reportado por el NE. En caso de no ser reportado el campo debe ir vacío.
edgePointRelated	boolean	TRUE en caso de ser un evento relacionado a un PTP que es edge point, FALSE en cualquier otro caso.
attributeList	notifications::NVList_T	Lista de los nombres de los atributos que fueron modificados y sus correspondientes valores nuevos.
remainder_of_body		
El objeto modificado según el tipo especificado en *objectType.* El objeto viaja dentro de un tipo Any de CORBA.		

- Para *type_name* NT_STATE_CHANGE la especificación MTNM sufrió cambios. El *remainder_of_body* fue modificado respecto a la especificación MTNM. Según MTNM debía ir en NULL sin embargo en el marco de nuestro proyecto se empleo para enviar el objeto modificado, de esta forma se simplificó la implementación enviando el cambio de estado en el propio objeto. Por lo anterior el campo *attributeList* queda obsoleto. Para una justificación mas detallada acerca de dicha decisión referirse a la sección 1.2 *Detalles de implementación*.

filterable_data		
Nombre	**Tipo de dato**	**Descripción**
notificationId	string	No están garantizadas la unicidad y la secuencia del notificationid.
objectName	globaldefs::NamingAttributes_T	Nombre del objeto cuyo estado fue modificado.
objectType	notifications::ObjectType_T	Tipo del objeto cuyo estado fue modificado.
emsTime	globaldefs::Time_T	Tiempo en el que el evento fue reportado en el sistema EMS.
neTime	globaldefs::Time_T	Tiempo reportado por el NE. En caso de no ser reportado el campo debe ir vacío.
edgePointRelated	boolean	TRUE en caso de ser un evento relacionado a un PTP que es edge point, FALSE en cualquier otro caso.
attributeList	notifications::NVList_T	Nombre de los atributos de estado y sus valores nuevos.
remainder_of_body		
El objeto modificado según el tipo especificado en *objectType.* El objeto viaja dentro de un tipo Any de CORBA.		

8.7.2 Detalles de implementación generales

Las notificaciones fueron implementadas utilizando un conjunto de clases de JacORB [18] referentes al servicio de notificaciones de CORBA junto con las clases MTNM relacionadas y el documento notificationServiceUsage.html que especifíca como deberán ser implementadas las notificaciones para MTNM.

Clases MTNM
globaldefs
notifications

Clases JacORB
CosNotification
CosNotifyChannelAdmin
CosNotifyComm
CosNotifyFilter
CORBA::Any

Las clases implementadas para nuestro proyecto relacionadas con notificaciones se encuentran en el paquete *notificaciones* y son las siguientes:

Clase	***Descripción***
notificaciones::NotifStructEvent	Representa un evento estructurado MTNM, el mismo toma como partida la definición de evento estructurado especificado por la OMG[10] y que JacORB[18] implementa.
notificaciones::EventTypeMTNM_T	Representa los distintos tipos de eventos MTNM según la especificación incluida en el documento de referencia notificationServiceUsage.html
notificaciones::NotifConsumidor	Representa la conexión al canal de eventos del lado de una aplicación (o componente) consumidor. Esta clase oculta los detalles de la conexión al canal de eventos, obtiene del canal un objeto admin consumidor, obtiene de este un proxy proveedor y lo relaciona con un objeto consumidor.
notificaciones::InterfaceNotifConsumidor	Interface que debe implementar una aplicación (o componente) consumidor para recibir eventos del canal.
notificaciones::NotifProveedor	Representa la conexión al canal de eventos del lado de una aplicación (o componente) proveedor. Esta clase oculta los detalles de la conexión al canal de eventos, obtiene del canal un objeto admin proveedor, obtiene de este un proxy consumidor y lo relaciona con un objeto proveedor.

Rol de las notificaciones en el sistema

Las fachadas de capa de elemento son proveedores de eventos y el Inventario de capa de red es un consumidor que utiliza las notificaciones para mantener actualizados los datos relacionados a los objetos de capa de elemento.

Un *componente consumidor* tiene las siguientes características:

Clases que importa:

- NotifStructEvent
- EventTypeMTNM_T
- NotifConsumidor

Interface que implementa:

- InterfaceNotifConsumidor

El componente Inventario abre una sesión con capa de elemento y a través de ella obtiene una referencia al canal de notificaciones del sistema. Luego se conecta al mismo utilizando el método `NotifConsumidor::connect()`.

Como el modelo de eventos recomendado por MTNM es *push* el Inventario recibe los eventos de forma pasiva y debe por lo tanto implementar la interface `InterfaceNotifConsumidor`. Cada vez que un evento llega al canal el método `pushStructuredEvent(StructuredEvent structuredEvent)` de la interface anterior es invocado. Luego el evento se procesa como un evento estructurado según la especificación MTNM utilizando las operaciones de la clase `NotifStructEvent` realizada para tales efectos.

Para cerrar la conexión con el canal el Inventario invoca el método `NotifConsumidor::shutdown()`.

Un *componente proveedor* tiene las siguientes características:

Clases que importa:

- NotifStructEvent
- EventTypeMTNM_T
- NotifProveedor

Los componentes MLSNMgr, EMSMgr y ManagedElement obtienen una referencia al canal de notificaciones del sistema en sus respectivos constructores. Luego se conectan al canal utilizando el método `NotifProveedor::connect()`. Para enviar un evento al canal un proveedor invoca el método `NotifProveedor::sendEvent(StructuredEvent eventoStruc)`. Previamente el objeto evento es creado utilizando las operaciones de la clase `NotifStructEvent`.

No se implementaron filtros y propiedades de QoS en las notificacios debido a que excedían el alcance definido para nuestro proyecto.

Detalles relacionados con la especificación MTNM de evento estructurado

Para enviar y recibir notificaciones los componentes utilizan las operaciones de la clase `NotifStructEvent` que les permite crear y procesar un evento estructurado según la especificación MTNM. Las operaciones se dividen en dos grupos, las que tienen el fin de completar los campos `filterable_data` y `remainder_of_body` del cuerpo del evento y las que dado un evento permiten procesar dichos campos para obtener la información que contienen. Además para el caso de crear, luego que se tienen todas las partes del evento se debe invocar el método `NotifStructEvent::crearEvento(...)` pasando como parámetros cada parte.

Una de las decisiones de implementación con respecto a este tema es la forma de enviar los cambios en los atributos y en el estado de los objetos para los eventos NT_ATTRIBUTE_VALUE_CHANGE y NT_STATE_CHANGE. En este caso no se siguió completamente la especificación MTNM debido a que la misma hacia mas pesada la implementación y no aportaba desde el punto de vista académico a nuestro proyecto. La siguiente es la justificación acerca de la decisión tomada.

Según MTNM para eventos de tipo NT_ATTRIBUTE_VALUE_CHANGE y NT_STATE_CHANGE el campo `remainder_of_body` del evento no se completa. Por otro lado especifíca que los atributos modificados deberán viajar en el campo `attributeList` del cuerpo `filterable_data` del evento. El campo `attributeList` es de tipo `notifications::NVList_T`, este tipo de dato es una lista de pares *<name,value>* donde *name* es de tipo `String` y contiene el nombre del atributo mientras que *value* es de tipo `Any` y contiene el nuevo valor del atributo.

Como cada objeto MTNM tiene un gran número de atributos, de tipos muy distintos se tendría que contemplar cada caso particular, debido a que para cargar un `Any` es necesario utilizar los métodos `insert()` y `extract()` de la clase Helper correspondiente al tipo de dato del atributo. Por ejemplo:

Tipo: `Topology_T`

Helper: `Topology_THelper`

Para cargar un atributo de tipo `Topology_T` en un objeto `Any` *y* posteriormente agregarlo al campo `attributeList`:

```
Topology_T tipoMLSN = Topology_T.TOPO_SINGLETON;
Any userAny = _orb.create_any();
Topology _THelper.insert(userAny, tipoMLSN );
```

Para consumir un objeto almacenado en un `Any` de la lista `attributeList` primero hay que conocer cual es el atributo y emplear el `Helper` adecuado:

```
if  (evento.attributeList [0].name==“subnetworkType”){
      Topology_T tipoMLSN=Topology _THelper.extract(evento.attributeList [0].value);
}else....
```

En resumen, implementar los pasos del ejemplo para todos los atributos de los objetos MTNM lleva un tiempo considerable y desde el punto de vista académico no aporta demasiado. Por otro lado la especificación MTNM sobre no enviar todo el objeto modificado tiene sentido en sistemas con gran número de objetos donde eventos muy pesados afectarían la performance. Sin embargo no es el caso del prototipo de nuestro proyecto.

8.8 Temas no especificados

8.8.1 Inicialización y configuración

Para un sistema de utilidad real en una aplicación de telecomunicaciones, sería interesante poseer una infraestructura que ofrezca las siguientes funcionalidades:

- Inicialización y monitoreo remoto de componentes.
  Una herramienta que permita levantar, dar de baja, y ver el estado de los componentes de forma remota.
- Gestión automática de fallos.
  Podría estar integrado con el servicio de ciclo de vida de CORBA de manera de, por ejemplo, permitir reiniciar automáticamente componentes que dejan de responder.
- Configuración remota.
  Permitir la modificación de parámetros de configuración de componentes, de forma remota y en tiempo de ejecución.

Debido al alcance acotado de este proyecto, se decidió desarrollar una infraestructura mínima, con el único fin de facilitar el desarrollo[50].

8.8.2 Seguridad

Como se detalla en la sección dedicada a la gestión de sesiones[51], las interfaces especificadas por MTNM no son suficientes para implementar un mecanismo de seguridad. En particular, no hay un mecanismo que les permita a las fachadas asociar a sus usuarios (clientes que invocan métodos) con sesiones abiertas en la *root interface*.

Posibles vías de investigación son el servicio de seguridad de CORBA, y la utilización de túneles SSL para el tráfico de CORBA.

El mecanismo de seguridad también tiene que abarcar al canal de notificaciones.

8.8.3 Persistencia

A continuación se presenta la lista de fachadas, sus correspondientes objetos, y requerimientos de persistencia.

- *Root interfaces* (EMSSession)
  Debido a las restricciones decididas para la gestión de sesiones[52], la fachada EMSSessionFactory no necesita mantener persistencia de objetos.
- ME, PTP, CTP(ManagedElementMgr)
  Esta fachada gestiona objetos que se corresponden a dispositivos físicos y a sus estados. Una fuente para esta información podría ser un componente de descubrimiento de la red. Sin embargo, eso está fuera del alcance de este proyecto. Sobre los detalles de la persistencia para este componente, ver la documentación del proyecto Gestión de Inventario (Gesinv).

50 Ver sección 8.1 Inicialización y configuración.

51 Ver sección 8.2 Sesiones.

52 Ver sección 8.2 Sesiones.

- MLSN y TL
  Hay métodos dadores para objetos de estas clases tanto en la fachada MLSNMgr como en la EMSMgr. Si bien la fachada responsable por estos objetos es MLSNMgr, queda más cómodo que la persistencia se realice en el componente EMSMgr,.y que e el MLSN Mgr la obtenga desde allí. Esto se debe al hecho que al componente EMSMgr se le agregó un interface CORBA específicamente para tareas de gestión[53]. Es importante hacer notar que este desplazamiento no compromete la iterface MTNM, ya que MTNM no provee método modificadores de estos objetos. Por lo tanto, el comportamiento es el mismo independientemente de si estos residen en un componente u otro.
- SNC
  Estos objetos deben ser almacenados de forma persistente, de forma de poder responder ante una reinicialización del componente[54].
- EMS
  MTNM no define métodos modificadores para este objeto, por lo que se agregó una interface para gestionarlos[55]. Debe ser mantenido de forma persistente.

El criterio general usado es que cada fachada se encargue de mantener la persistencia de sus propios objetos, con la excepción indicada para los MLSN y TL. La persistencia se implementa en archivos XML[56].

8.8.4 Transacciones

Para la gestión de fallos al realizar operaciones de configuración de dispositivos, MTNM utiliza gestión explícita de estados. Por ejemplo, una SNC pasa por varios estados durante su activación[57]. Una SNC para activarse necesita reservar varios recursos (cross connect), y estos recursos pueden estar ocupados por otros SNCs. Al fallar una activación de SNC, los estados de los objetos involucrados deben ser analizados explícitamente para decidir cursos de acción (por ejemplo, para deshacer lo que ya se hizo).

De aquí surgen los riesgos típicos inherentes al acceso concurrente a recursos compartidos, tales como bloqueos mutuos, violaciones a la integridad de los datos, e inanición.

Probablemente se lograría una simplificación importante de la interfaz si se tuviera en cuenta el concepto de transacción, tal como se lo entiende en el mundo de las bases de datos. De este modo, la implementación de todos los Cross connects de un SNC podría realizarse como si fuera una acción atómica, que en caso de fallar dejaría los dispositivos en el mismo estado que estaban al principio.

Esta es una dirección de estudio que nos parece sería interesante continuar.

53 Ver sección 8.5 EMS Manager.

54 Ver sección 8.6 MultiLayerSubnetwork Manager.

55 Ver sección 8.5 EMS Manager.

56 Ver Apéndice II: XML/XSD.

57 Ver documento de MTNM TMF513V2_0.pdf.

8.9 Cliente de capa de elemento: emsGUI

La emsGUI esta diseñada de manera de permitir al usuario realizar ciertas tareas en la capa de elemento (EMS) de forma amigable. Para realizar lo anterior fueron seleccionados algunos casos de uso relevantes sobre los que se definió la arquitectura, que en su mayoría son un mapeo directo a los casos de uso definidos por MTNM en su estándar.

El usuario podrá, a través de esta GUI, abrir una sesión con la capa de elemento y una vez abierta, realizar tareas de administración como crear conexiones (SNC), eliminarlas, activarlas, etc, y además poder navegar en forma gráfica los objetos que se encuentran en los distintos componentes de capa de elemento.

Es importante dejar claro en este punto que ésta GUI está hecha principalmente para poder testear las fachadas y todo lo que compone al EMS. Por esta razón no fue hecho un trabajo muy profundo en lo que respecta a la performance de la misma. Esta GUI usa CORBA para realizar sus tareas, lo que implica un tráfico importante en el momento que el usuario inicia una sesión y comienza a desarrollar tareas en ella, principalmente al navegar los objetos que se muestran en forma de árbol. Una manera de mejorar esto es implementar una especie de caché y manejar las notificaciones para manipular los objetos en esos casos.

Se enumeran a continuación las funcionalidades de la GUI[58] :

1) Iniciar una sesión con el EMS.
2) Cerrar una sesión con el EMS.
3) Setear el Owner de un objeto MTNM.
4) Setear el User Label de un objeto MTNM.
5) Setear el Native EMS Name de un objeto MTNM.
6) Obtener los datos de un objeto MTNM.
7) Crear una SNC.
8) Activar una SNC.
9) Desactivar una SNC.
10) Eliminar una SNC.
11) Crear y activar una SNC.
12) Desactivar y eliminar una SNC.

Para conocer más sobre emsGUI referirse al Apéndice VII: *Manual de usuario EMS GUI*.

58 Ver detalles de los casos de uso en el Apéndice I: Documentación de Capa de Elemento.

9 CAPA DE RED

9.1 CaSMIM

9.1.1 Problema de negocio

El problema central en que se enfoca CaSMIM es poder ofrecer servicios de conectividad con parámetros de calidad (QoS), de forma ágil, segura y a un costo razonable. Un requerimiento de un único servicio puede implicar que muchas *connections*[59] sean establecidas usando múltiples tecnologías. Cada una de estas múltiples tecnologías puede a su vez ser implementada con equipamiento de múltiples vendedores.

Si recurrimos a procesos manuales o enviamos mensajes distintos según la interface cliente de cada uno de los *management systems* podemos tardar un tiempo considerable desde que se inicio la solicitud hasta que el nuevo servicio fue creado, y un gran gasto entre la operativa y el desarrollo del sistema.

Problemas adicionales introduce el requerimiento relacionado al manejo de fallas en los servicios debido a que es tradicional enfocar la gestión de fallas asociada al equipo y no al servicio. Para asociar las fallas a los servicios hay que juntar todas las piezas de información a través de todos los sistemas involucrados.

9.1.2 Escenarios soportados

Escenarios principales desde el punto de vista funcional.

- Escenario de negocio 1: *Aprovisionamiento de servicios de conectividad end-to-end.*

  Descripción: El escenario de negocio principal es el aprovisionamiento de servicios de conectividad en ambientes multi-vendedores y multi-tecnologías. Un variedad de tecnologías puede ser posible, como por ejemplo las llamadas tecnologías orientadas a la conexión como ATM, SONET/SDH, WDM y Frame Relay y también las llamadas tecnologías de acceso como ADSL y HFC.

  Principios: El aprovisionamiento de servicios de conectividad consiste en lo siguiente:

- El descubrimiento de la información topológica de la red. Este es un requerimiento fuerte para el *Service Provider.* Dicho requerimiento elimina el trabajo manual de cargar la topología de la red en los Network Management Systems.
- El descubrimiento de las capacidades de la red: las *features*[60] soportadas.
- La creación de *connections* para soportar servicios.
- La modificación de *connections* en respuesta a requerimientos de modificación de servicios.
- La eliminación de *connections*.
- Descubrimiento/Consulta de rutas usadas por las c*onnections*.
- Restricciones de ruteo: debe ser posible usar restricciones de ruteo al establecer una *connection*.
- Escenario de negocio 2: *Correlación de fallas de red a servicios de conectividad.*

  Descripción: El otro escenario de negocio es el mapeo de alarmas en la red a notificaciones comprensibles tanto para gestión de servicios de alto nivel para clientes (sms), como para sistemas de gestión de *connection* (nms).

59 Ver Apéndice III: Documentación de Capa de Red, sección 1.Términos clave.

60 Ver Apéndice III: Documentación de Capa de Red, sección 1.Términos clave.

Principios: Este escenario solo cubre las alarmas que afectan los servicios. No asume que algún Sistema Servidor sea un sistema de gestión de fallas con funcionalidades complejas como correlación de fallas. Este solo requiere que el *Servidor* sea capaz de identificar una falla en una *connection*, y transmitir la información usando la terminología de servicios. Una *connection* para un Sistema Servidor aparece como un *servicio de conectividad* brindado a un Sistema Cliente[61].

Nuestro proyecto solo cubre el escenario de negocio 1: *Aprovisionamiento de servicios de conectividad end-to-end* con algunas restricciones en los temas de descubrimiento de *features* soportadas y decisiones y restricciones de ruteo. Por mas detalles ver sección *9.1.3 Alcance.*

9.1.3 Alcance

La interface CaSMIM cubre las interfaces entre un SMS y un sistema subyacente que ofrece servicios de gestión de *connections*. La capa de gestión de red puede manejar múltiples NMSs, o sea múltiples *Domain Managers*[62]. Sevicios de conectividad end-to-end pueden ser soportados entre *Domain Managers*. Esta interface también puede ser usada directamente entre una aplicación con funcionalidades de capa de servicio como por una aplicación con funcionalidades de capa de elemento. Por ejemplo, cuando un ISP (Internet Service Provider) gestiona servicios para provee a sus clientes, puede emplear una interface directa entre la aplicación de gestión de servicios y la aplicación de gestión de elemento, gestionando el dispositivo final como una o más *connections*.

En el marco de nuestro proyecto no se consideró la posibilidad de usar la interface CaSMIM para comunicar aplicaciones de capa de servicio con capa de elemento. La interface quedó restringida a la comunicación entre capa de servicio y capa de red y entre componentes de capa de red entre si.

9.1.4 Requerimientos

Según CaSMIM se deberá implementar una interface genérica que ofrezca un mecanismo simple para ofrecer servicios de conectividad y monitorearlos durante todo su ciclo de vida. Para ello se deberá cubrir una serie de requerimientos algunos de los cuales no entran dentro del alcance de nuestro proyecto. Para conocer detalles acerca de los requerimientos cubiertos y no cubiertos ver Apéndice III: *Documentación de Capa de Red* el cual referencia al documento TMF 508 Connection and Service Management Information Agreement, en su versión v30_pe[63]. También se podrá encontrar en dicho apéndice los casos de uso especificados por CaSMIM relacionados a los requerimientos cubiertos y adaptados según el alcance de nuestro proyecto.

9.2 miCasmim

El conjunto de IDLs miCasmim es un subconjunto de la interface CaSMIM definida por el TMF en el documento TMF 807 versión 1.5 Junio del 2001, CORBA IDL Solution Set[64].

MiCasmim se especificó de forma de servir como interface a un componente Provisioning en el marco de nuestro proyecto. La misma incluye entre otras, tres interfaces principales: Subnetwork2, Connection2 y Termination2. Dichas interfaces son fachadas de granularidad gruesa por clase ya que ofrecen acceso a una colección de objetos subnetwork, connection y termination llamados objetos de segunda clase. A su vez Connection2 y Termination2 heredan de la interface ManagedObject2.

61 Ver definiciones de Sistema Servidor y Cliente en el Apéndice III: Documentación de Capa de Red, sección 1.Términos clave.

62 Ver Apéndice III: Documentación de Capa de Red, sección 1.Términos clave.

63 Ver documento de CaSMIM TMF508_v30_pe.pdf.

64 Ver Apéndice IV: IDLs miCasmim.

9.3 Provisioning

El componente Provisioning de capa de red es el responsable de la gestión de los servicios de conectividad. El mismo ofrece a sus clientes tres interfaces: Subnetwork2, Connection2 y Termination2 especificadas en el conjunto de IDLs miCasmim.

Por otro lado dicho componente usa las interfaces del componente Inventario de capa de red: Iinventory_I, Iconnection_I, TreeViewObject_I y las interfaces de capa de elemento: EmsSessionFactory_I, NmsSession_I, MultiLayerSubnetworkMgr_I. Se presenta a continuación un diagrama con las interfaces mencionadas.

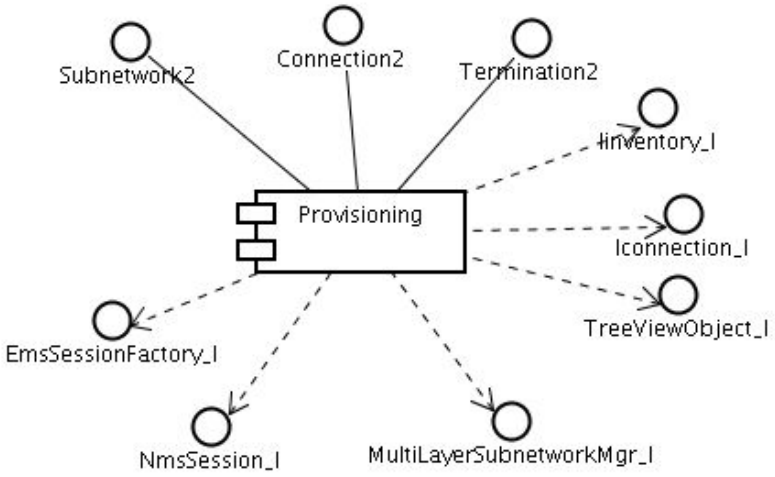


Diagrama 36 Contexto del componente Provisioning

El componente Provisioning es levantado y puesto a disposición de sus clientes al instanciar y publicar en el servicio de nombres de CORBA tres componentes mas pequeños que representan cada una de sus interfaces. A continuación se presentan comentarios relacionados al análisis y diseño de cada uno, asi como también detalles de implementación.

Para acceder a las definiciones de los términos utilizados en capa de red (connection, termination, etc) referirse a la sección *Términos clave* en el apéndice *Documentación Capa de Red.*

9.4 Subnetwork2

9.4.1 Análisis y diseño

Alcance

El componente Subnetwork2 en el marco de nuestro proyecto cubre las siguientes funcionalidades:

- crear una connection en estado EMPTY, guardarla en el Inventario y asignarle un nombre miCasmim mediante el método `create_connection2(...)`.
- crear una conection en estado RESERVED y asignarle los parámetros deseados mediante el método `create_connection_populated2(...)`.
- obtener la lista de nombres de connection existentes en la subnetwork mediante el método `get_contained_connections()`.

Inicialización

Al instanciar un componente Subnetwork2, el mismo recibe en su constructor una lista de parámetros necesarios para su correcta inicialización. Como parte de la inicialización el componente accede a las interfaces del componente Inventario y accede a la interface de la fachada Connection2 quien gestiona sus connections. Los parámetros son los siguientes:

Parámetro	*Tipo*	*Descripción*
java_class	String	Es una etiqueta que indica al Loader la identidad del componente a levantar (en este caso, el Subnetwork2)
ns_name	String	El nombre con el que se quiere que se registre el componente en el servicio de nombres de CORBA.
SubnetworkName	String	Nombre del componente Subnetwork2 en el Servicio de Nombres. Este parámetro es utilizado para construir nombres de connection según el formato miCasmim[65].
Connection2	String	Nombre de la interface Connection2 del componente Provisioning en el servicio de nombres de CORBA .
IInventory	String	Nombre de la interface IInventory del componente Inventario en el servicio de nombres de CORBA .
IConnection	String	Nombre de la interface IConnection del componente Inventario en el servicio de nombres de CORBA .
TreeViewObject	String	Nombre de la interface TreeViewObject del componente Inventario en el servicio de nombres de CORBA .

Tabla 8 Parámetros de Subnetwork2

En el constructor se obtiene una referencia a cada una de las interfaces del Inventario y una referencia a la Interface Connection2 del componente Provisioning.

65 Ver sección 9.7.1 NombreMiCasmim.

Persistencia

El componente Subnetwork2 no gestiona objetos directamente, una vez creadas las connections son gestionadas por la fachada Connection2 y cuando Subnetwork2 necesita acceder a su lista de connections las obtiene del componente Inventario.

9.4.2 Detalles de implementación

La implementación del componente tiene las siguientes generalidades:

- Utiliza una única referencia a la interface Connection2 durante todo su ciclo de vida. El mismo comentario aplica para las interfaces del Inventario.
- Utiliza la clase NombreMiCasmim para asignar nombres a sus connections y para facilitar la manipulación de los tipos ConnectionIdentities y TerminationIdentities.
- Utiliza la clase MiCasmimDefs para unificar las definiciones de constantes como por ejemplo: estados de una connection, tipos de errores reportados, etc. Además usa los métodos disponibles para construir nombres de objetos según la especificación miCasmim.

Diagramas de secuencia

Los diagramas de secuencia muestran la interacción de la interface Subnetwork2 de Provisioning con los restantes componentes de capa de red y con capa de elemento.

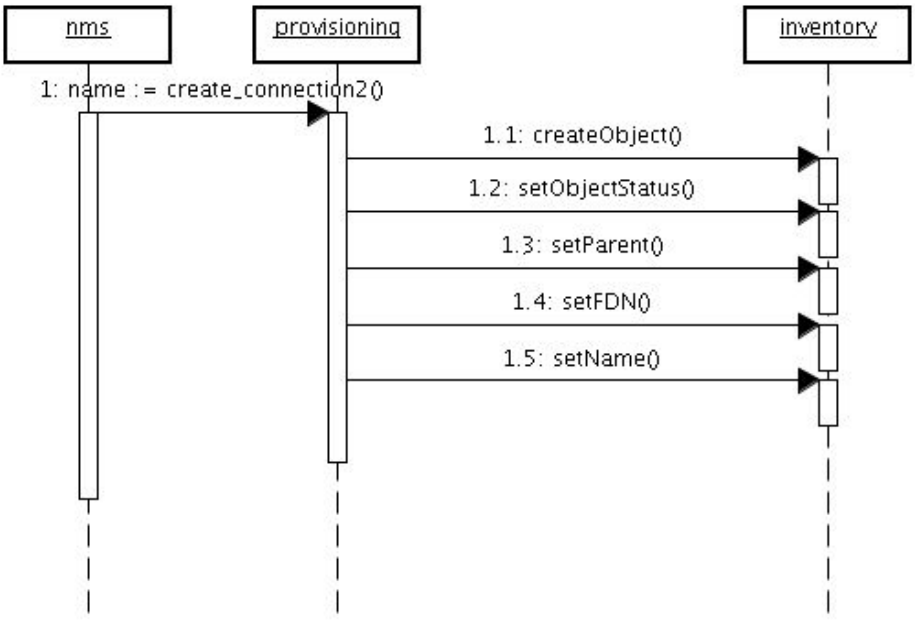


Diagrama 37 Crear una connection en estado EMPTY

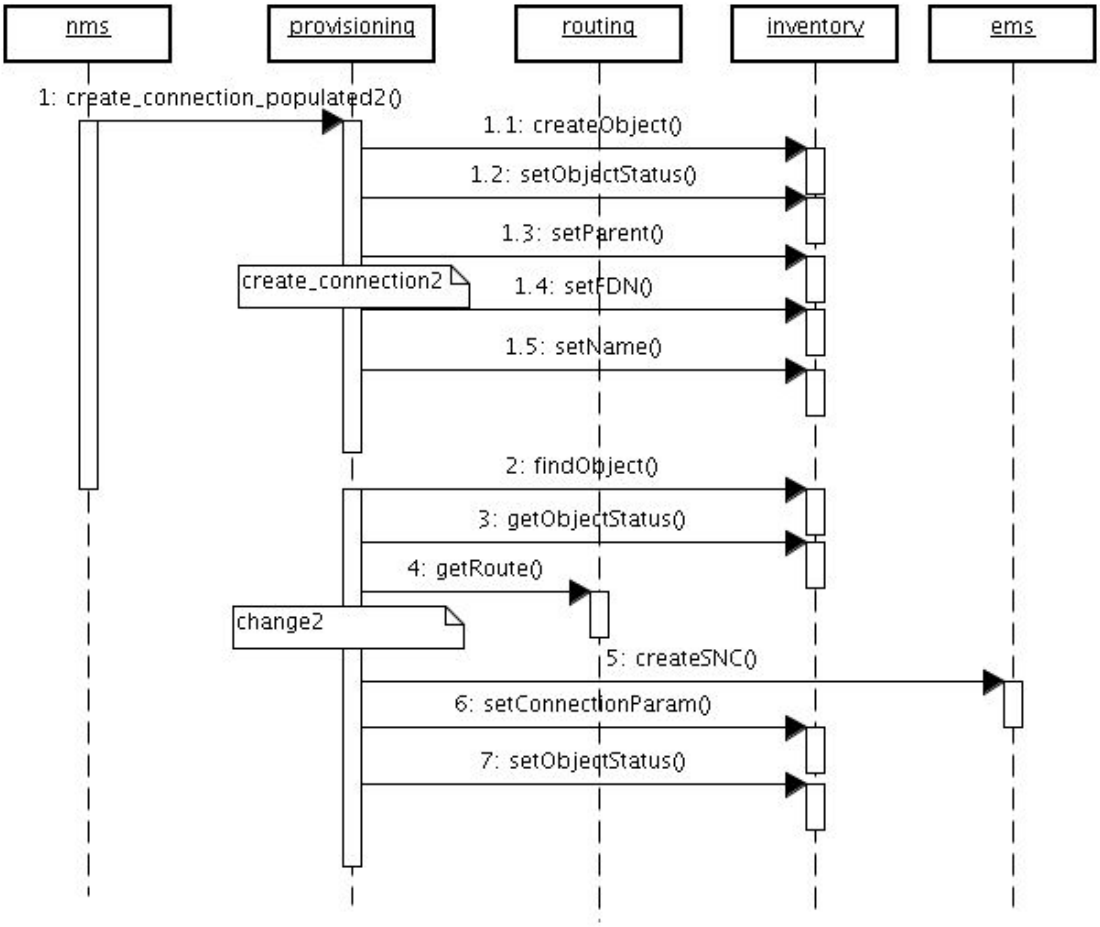


Diagrama 38 Crear una connection en estado RESERVED

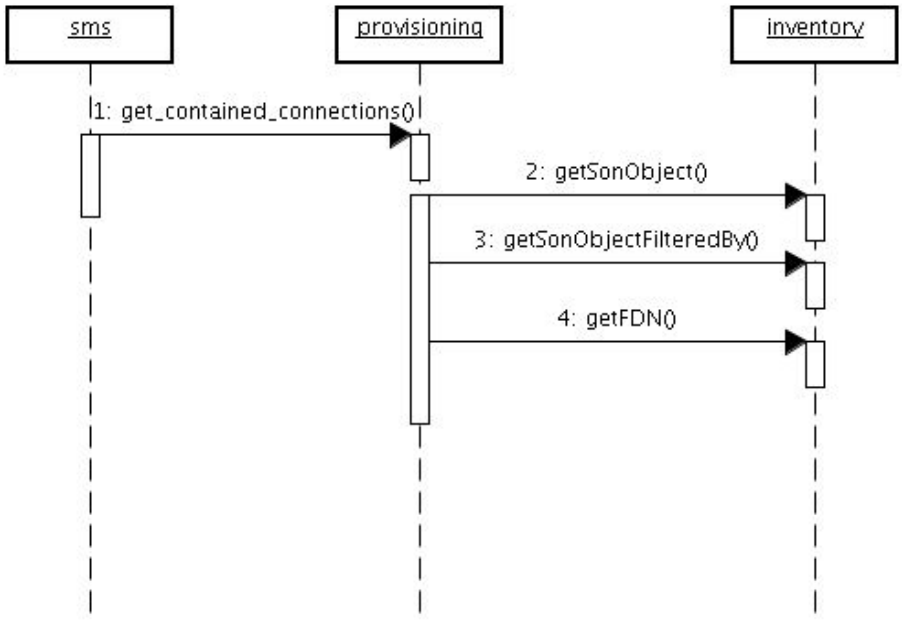


Diagrama 39 Obtener la lista de connection de la subnetwork

Estructura de datos

Subnetwork2 tiene un conjunto de variables globales que mantienen las referencias a las interfaces que el componente necesita durante su operativa.

Además utiliza un hash para almacenar los parámetros necesarios que recibe en su constructor, los mismos fueron detallados en la sección *Inicialización*:

```
private Map _parametros
```

Para ver detalles relacionados a las operaciones públicas y privadas implementadas referirse al Apéndice III: *Documentación de Capa de Red.*

9.5 Connection2

9.5.1 Análisis y diseño

Alcance

El componente Connection2 en el marco de nuestro prototipo cubre las siguientes funcionalidades:

- recuperar los atributos de uno o mas objetos connection identificados por sus nombres mediante el método `get_connections_attributes2(...)`,
- cambiar el estado de una connection existente identificada por su nombre mediante el método `change2()`
- eliminar una connection existente identificada por su nombre mediante el método `remove2()`.

Inicialización

Al instanciar un componente Connection2, el mismo recibe en su constructor una lista de parámetros necesarios para su correcta inicialización. Como parte de la inicialización el componente abre una sesión con capa de elemento y accede a las interfaces del componente Inventario. Los parámetros son los siguientes:

Parámetro	*Tipo*	*Valor por defecto*	*Descripción*
java_class	String		Es una etiqueta que indica al Loader la identidad del componente a levantar (en este caso, el Connection2)
ns_name	String		El nombre con el que se quiere que se registre el componente en el servicio de nombres de CORBA.
emsSessionFactoryName	String		Nombre del EmsSessionFactory en el Servicio de Nombres. Connection2 accede al EmsSessionFactory para iniciar una sesión con capa de elemento.
emsUser	String		Usuario para la sesión con capa de elemento.
emsPasswd	String		Password para la sesión con capa de elemento.
SubnetworkName	String		Nombre del componente Subnetwork2 en el Servicio de Nombres. Este parámetro es necesario ya que uno de los atributos de los objetos *connections* gestionados por Connection2 es el IOR de la Subnetwork que obtiene del servicio de nombres de CORBA.
IInventory	String		Nombre de la interface IInventory del componente Inventario en el servicio de nombres de CORBA.
IConnection	String		Nombre de la interface IConnection del componente Inventario en el servicio de nombres de CORBA .
TreeViewObject	String		Nombre de la interface TreeViewObject del componente Inventario en el servicio de nombres de CORBA.
Routing	String		Nombre de la interface Routing2 del componente Routing en el servicio de nombres de CORBA.
intervaloAgenda	Integer	1000	Cada cuantos milisegundos se revisa la lista de SNCs pendientes para ser asociadas a sus correspondientes connections en el Inventario.

Tabla 9 Parámetros de Connection2

El constructor inicia una sesión con capa de elemento y posteriormente obtiene una referencia a la fachada MLSNMgr con el objetivo de ejecutar operaciones sobre los objetos SNCs que dicha fachada gestiona. Además obtiene una referencia a cada una de las interfaces del Inventario y una referencia a la Interface Routing2 del componente Routing. Para finalizar activa el control de la lista de SNCs agendadas para ser asociadas a sus correspondientes connections en el Inventario, inicialmente esta lista esta vacía.

Persistencia

El componente Connection2 mantiene todos sus objetos persistentes en el componente Inventario.

Diagrama de Estados

Durante su ciclo de vida una connection pasa por distintos estados que se describen a continuación. Al crear un objeto connection este pasa a existir en el Inventario y su estado es EMPTY.

En estado EMPTY el objeto no tiene ninguno de sus parámetros de conexión (common features y temination features) inicializados y por lo tanto no brinda todavía un servicio de conectividad.

Un objeto en estado EMPTY debe ser inicializado y para ello se debe determinar la ruta de SNCs que lo constituyen. Una vez determinada la ruta se crean las SNCs correspondientes y se inicializan los parámetros de conexión en el Inventario. Como resultado el estado del objeto sera IMPLEMENTED.

El estado IMPLEMENTED indica que el objeto esta en proceso de inicialización el cual termina cuando todas las SNCs que lo componen han sido relacionadas con él en el Inventario y el objeto pasa a estado RESERVED.

En estado RESERVED una connection esta disponible para prestar un servicio de conectividad y esta pronta para ser activada. En el momento que una connection es activada todas las SNCs que la forman son activadas y la connection pasa a estado OPERATIONAL brindando un servicio de conectividad a un usuario determinado.

En estado OPERATIONAL una connection puede ser desactivada y pasar nuevamente a estado RESERVED, esta operación implica desactivar cada una de las SNCs que la forman.

El ciclo de vida de una connection finaliza cuando todas las SNCs relacionadas son desactivadas y borradas y la connection deja de existir en el Inventario, momentáneamente la connection pasa a estado REMOVED. Es posible eliminar una connection cuando esta en estado OPERATIONAL, para ello previamente pasa a estado RESERVED donde todas las SNCs relacionadas son desactivadas. Estando en estado RESERVED todas las SNCs son borradas y posteriormente el objeto connection deja de existir en el Inventario.

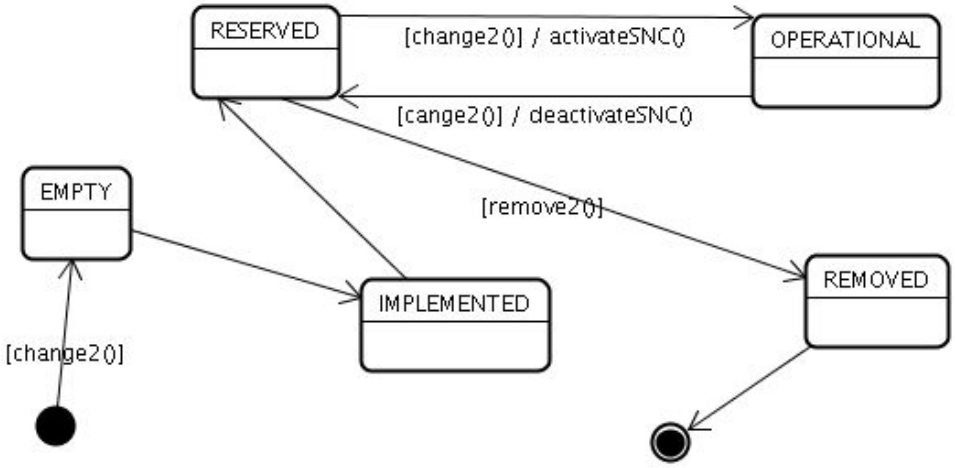


Diagrama 40 Diagrama de estado de una connection

9.5.2 Detalles de implementación

La implementación del componente tiene las siguientes generalidades:

- Utiliza un única referencia al MLSNMgr durante todo su ciclo de vida. El mismo comentario aplica para las interfaces del Inventario y del Routing.
- Utiliza la clase NombreMiCasmim para facilitar la manipulación de los tipos ConnectionIdentities y TerminationIdentities.
- Utiliza la clase MiCasmimDefs para unificar las definiciones de constantes como estados de una connection, tipos de errores reportados, etc. Además usa los métodos disponibles para construir nombres de objetos según la especificación miCasmim.

Diagramas de secuencia

Los diagramas de secuencia muestran la interacción de la interface Connection2 de Provisioning con los restantes componentes de capa de red y con capa de elemento.

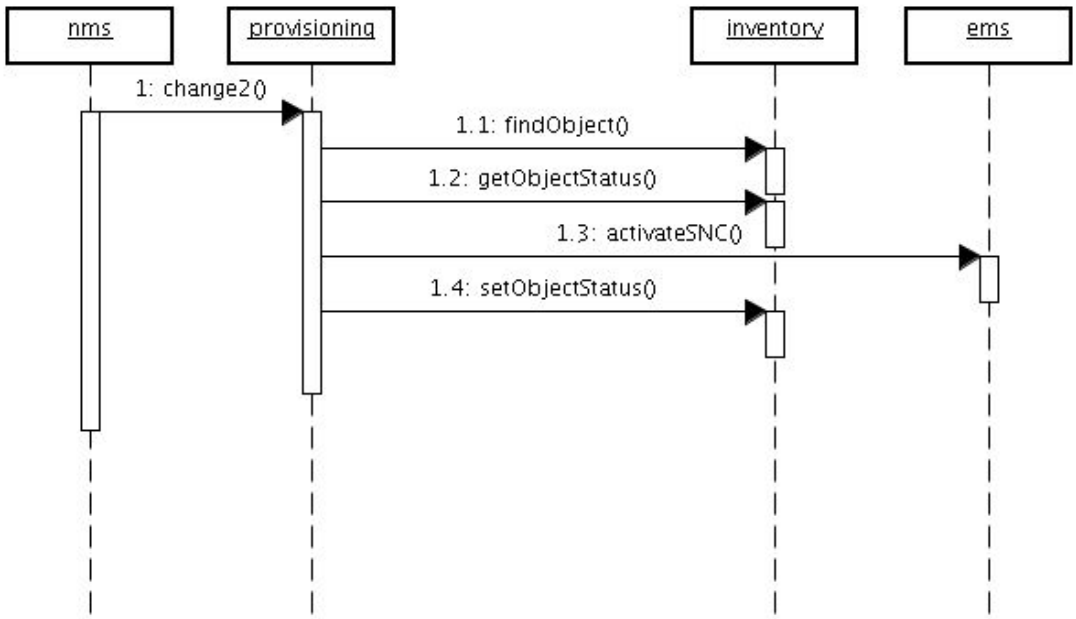


Diagrama 41 Activar una connection

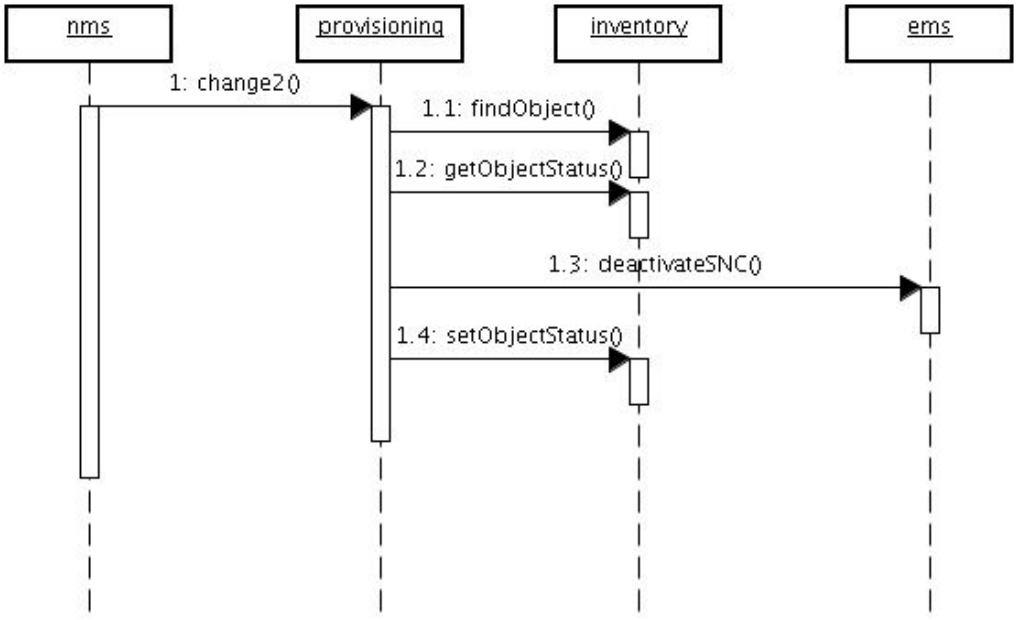


Diagrama 42 Desactivar una connection

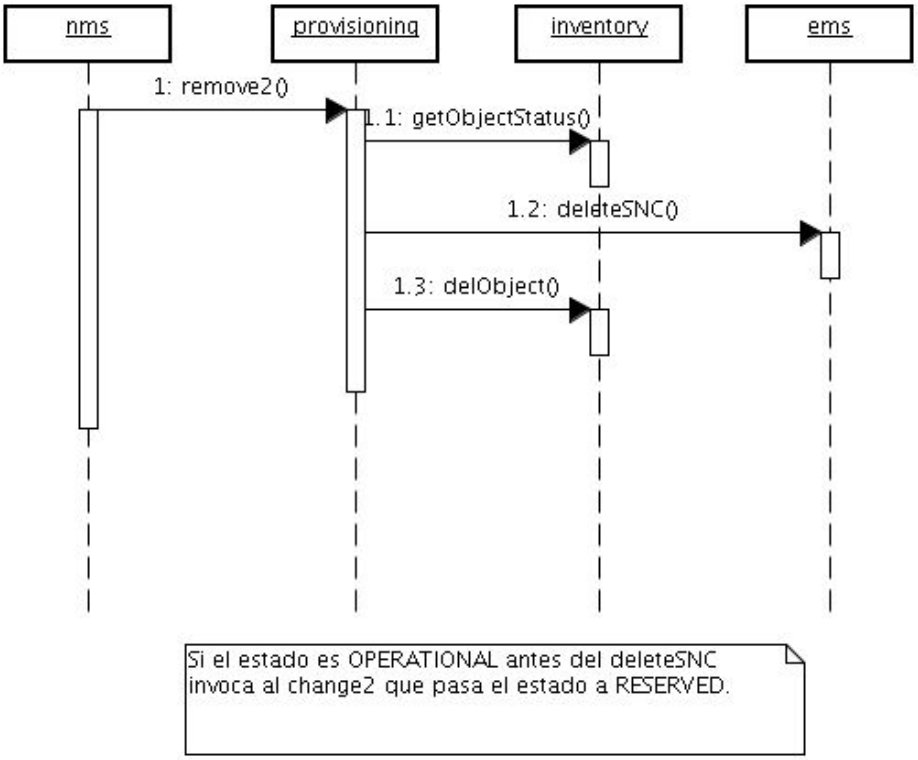


Diagrama 43 Eliminar una connection

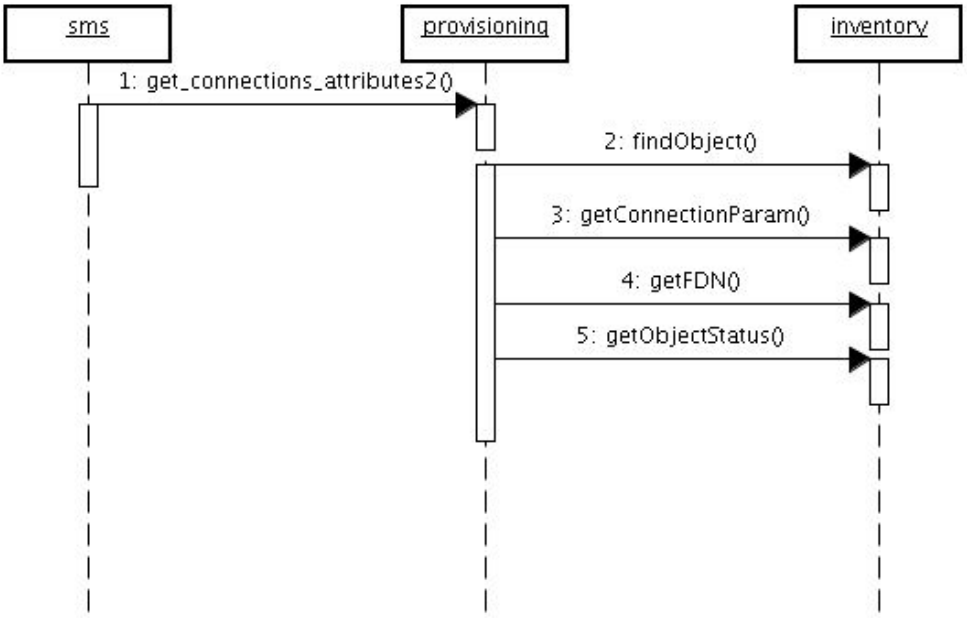


Diagrama 44 Obtener los atributos de una connection

Estructura de datos

Connection2 tiene un conjunto de variable globales que mantienen las referencias a las interfaces que el componente necesita durante su operativa. Además tiene la siguiente estructura de datos:

```
private Map sncsPendientes=Collections.synchronizedMap(new HashMap());
```

La variable global `sncsPendientes` es un hash donde la clave es un connID (entero que identifica una connection en el Inventario) y el contenido de la cubeta es una lista de SNCs relacionadas a la connection correspondiente que fueron agendadas con el objetivo de ser asociadas a la misma en el Inventario.

Esta estructura es necesaria para resolver lo siguiente: como parte de la inicialización de una nueva connection se solicita a capa de elemento crear una o mas SNCs y posteriormente se asocian a la connection en el Inventario. Una vez creadas las SNCs aparecen en el Inventario como resultado de notificaciones enviadas por el MLSNMgr. Estas notificaciones pueden tardar en llegar y por lo tanto puede pasar que las SNCs no estén disponibles en el Inventario en el momento de inicializar la connection y sea necesario intentar mas tarde.

El procedimiento `ChequearSncsPendientes()` es ejecutado cada un determinado intervalo en milisegundo configurable[66], el mismo recorre secuencialmente la lista de connID (lista de claves del hash) de la estructura de datos `sncsPendientes` . Para cada connID intenta: asociar el connID con la lista de SNCs almacenadas en la cubeta correspondiente y en caso de lograr que la lista quedara vacía, cambia el estado de la connection a RESERVED.

La estructura `sncsPendientes` y el acceso a la misma esta sincronizado para preveer accesos concurrentes como por ejemplo, la llega de una nueva solicitud de inicializar una connection que necesita agregar una nueva entrada a `sncsPendientes` justo en el momento en que se está ejecutando `ChequearSncsPendientes()`.

Por último el componente utiliza un hash para almacenar los parámetros necesarios que recibe en su constructor los mismos fueron detallados en la sección *Inicialización*:

```
private Map _parametros
```

Para ver detalles relacionados a las operaciones públicas y privadas implementadas referirse al Apéndice III: *Documentación de Capa de Red.*

Mapeo de datos miCasmim vs Inventario

Para almacenar los parámetros de una connection (common features y termination features) en el componente Inventario fue necesario definir un formato capaz de representar cualquier parámetro como un par <nombre, valor>. Si bien en el Inventario existen una serie de parámetros tabulados como por ejemplo: *servicetime*, *reliability*, *bandwidth* se prefirió dar a todos los parámetros un tratamiento genérico. De ésta forma la lista de common features y termination features puede ser variable. Para determinada connection la lista de common features puede incluir: *Protocol*, *Direction*, *Committed bandwidth,* mientras que para otra se puede agregar además el tiempo promedio entre fallas (*MTBF*) sin necesidad de modificar la implementación.

El Inventario permite mediante el método `setConnectionParam()` guardar cualquier información siempre que sea posible enviarla en `org.tmforum.mtnm.globaldefs.NameAndStringValue_T[]`. Un `NameAndStringValue_T` es un par de Strings donde uno es el nombre y el otro es el valor asociado a ese nombre.

66 Ver sección 9.5.1 Análisis y diseño, en particular Inicialización.

A continuación se muestra con un ejemplo la forma de generar el campo nombre para cada uno de los parámetros. Para facilitar la tarea se creo la clase MiCasmimDefs donde se definieron un conjunto de constantes[67].

Ejemplo:

connection: X

common features: *Committed bandwidth* = 5000, *Error rate* = 3

termination features:

termination id = 25, *Direction* = *source*, *Committed bandwidth* = 5000

termination id = 14, *Direction* = *sink*, *Committed bandwidth* = 5000

Pares <nombre, valor> correspondientes al ejemplo. En este caso el punto es el separador aunque puede ser cualquier otro caracter o caracteres dependiendo de la constante SEP definida en la clase MiCasmimDefs:

Nombre	***Valor***
PAR_C_CF.Committed bandwidth	5000
PAR_C_CF.Error rate	3
PAR_C_TID.25	25
PAR_C_TF.25.Direction	source
PAR_C_TF.25.Committed bandwidth	5000
PAR_C_TID.14	14
PAR_C_TF.14.Direction	sink
PAR_C_TF.14.Committed bandwidth	5000

La etiqueta PAR_C_TID es necesaria sobre todo para los casos donde no se desee especificar por lo menos una termination features a un termination. El identificador del termination de todas formas debe asociarse a la connection ya que forma parte de su especificación.

67 Ver seccón 9.7.2 MiCasmimDefs.

9.6 Termination2

9.6.1 Análisis y diseño

Alcance

El componente Termination2 en el marco de nuestro prototipo cubre la siguiente funcionalidad: devuelve la lista de nombres de terminations existentes en la subnetwork mediante el método `get_members()`. A traves de éste componente un aplicación cliente puede desplegar la lista de terminations donde es posible establecer un servicio de conectividad.

Inicialización

Al instanciar un componente Termination2, el mismo recibe en su constructor una lista de parámetros necesarios para su correcta inicialización. Como parte de la inicialización el componente accede a las interfaces del componente Inventario. Los parámetros son los siguientes:

Parámetro	***Tipo***	***Descripción***
java_class	String	Es una etiqueta que indica al Loader la identidad del componente a levantar (en este caso, el Termination2)
ns_name	String	El nombre con el que se quiere que se registre el componente en el servicio de nombres de CORBA.
SubnetworkName	String	Nombre del componente Subnetwork2 en el Servicio de Nombres. Este parámetro es utilizado para construir nombres de terminations según el formato miCasmim[68].
IInventory	String	Nombre de la interface IInventory del componente Inventario en el servicio de nombres de CORBA .
IConnection	String	Nombre de la interface IConnection del componente Inventario en el servicio de nombres de CORBA .
TreeViewObject	String	Nombre de la interface TreeViewObject del componente Inventario en el servicio de nombres de CORBA .

Tabla 10 Parámetros de Termination2

En el constructor se obtiene una referencia a cada una de las interfaces del Inventario.

Persistencia

El componente Termination2 no crea nuevos objetos terminations y cuando necesita acceder a su lista de terminations los obtiene del componente Inventario.

68 Ver sección 9.7.1 NombreMiCasmim.

9.6.2 Detalles de implementación

La implementación del componente tiene las siguientes generalidades:

- Utiliza una única referencia a las interfaces del Inventario durante todo su ciclo de vida.
- Utiliza la clase NombreMiCasmim para asignar nombres a sus terminations y para facilitar la manipulación del tipo TerminationIdentities.
- Utiliza la clase MiCasmimDefs para unificar las definiciones de constantes como por ejemplo para los tipos de errores reportados. Además usa los métodos disponibles para construir nombres de objetos terminations según la especificación miCasmim.

Diagramas de secuencia

Los diagramas de secuencia muestran la interacción de la interface Termination2 de Provisioning con los restantes componentes de capa de red y con capa de elemento.

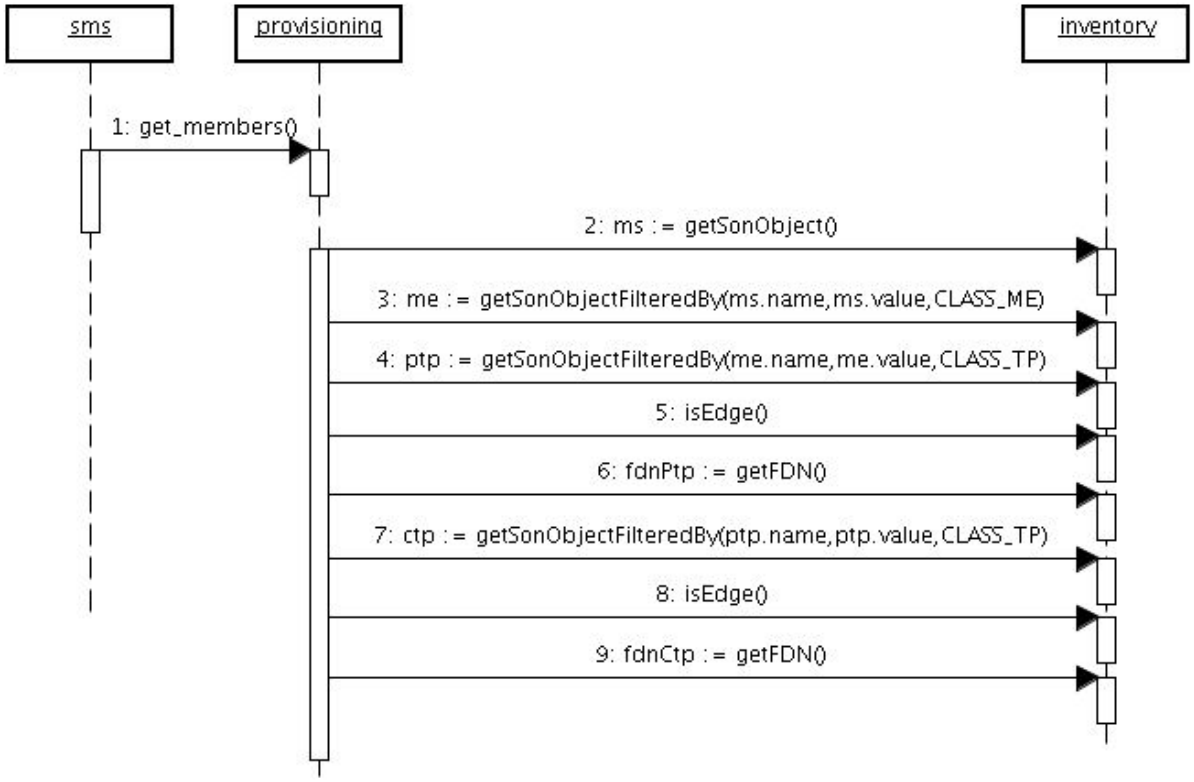


Diagrama 45 Obtener la lista de nombres de terminations

Estructura de datos

Termination2 tiene un conjunto de variables globales que mantienen las referencias a las interfaces que el componente necesita durante su operativa.

Además utiliza un hash para almacenar los parámetros necesarios que recibe en su constructor, los mismos fueron detallados en la sección *Inicialización*:

```
private Map _parametros
```

Para ver detalles relacionados a las operaciones públicas y privadas implementadas referirse al Apéndice III: *Documentación de Capa de Red.*

9.7 Clases utilitarias

Las clases utilitarias se crearon con el objetivo de ser usadas por todos los componentes de manera de unificar conceptos y evitar el re-trabajo.

9.7.1 NombreMiCasmim

La clase `NombreMiCasmim.java` incluida en el paquete `mitiNum.nms.provisioning.miCasmimImpl` es la definición de la estructura de nombres jerárquica de un objeto de capa de red que fue especificada para nuestro proyecto.

El nombre de un objeto en miCasmim es una lista de pares *<name,value>.* El campo *name* es un String que varia según el tipo de objeto (especificado en la clase MiCasmimDefs) y el campo *value* es un String con el nombre del objeto. A modo de ejemplo se lista el nombre de una connection. El campo *name* igual a *ID* es común a los objetos connections y terminations y su correspondiente campo *value* representa el identificador del objeto en el Inventario. Para las connections el campo *value* se construye concatenando el String `Conn_` con el tiempo en milisegundos en que el objeto fue creado.

```
name="SUBNETWORK";value="SN2"
name="CONNECTION";value="Conn_1079485359498"
name="ID";value="4"
```

La clase `NombreMiCasmim` implementa la función `toString()` que genera un String a partir de un array de `NameComponent`. El formato del String fue especificado por nosotros y es el siguiente: *name* correspondiente al objeto seguido por SEPARADOR, concatenado con los sucesivos *value* que forman el nombre, separados por SEPARADOR. El ejemplo anterior quedaría, tomando como SEPARADOR el punto:

```
CONNECTION.Conn_1079485359498.4
```

Un detalle importante de `NombreMiCasmim` es que se implementó como una clase *hashable* (implementa las operaciones `hashCode()` y `equals()`) lo que permite que objetos de esta clase puedan ser agregados a estructuras de hash dando la posibilidad de búsquedas eficientes.

Estructura de datos

Para brindar mayor flexibilidad se dejo pública la estructura ya que puede ser útil poderla cargar directamente.

```
public NameComponent[] value= new NameComponent[0];

public static String SEPARADOR=".";
```

La clase incluye tres constructores, el por defecto, un segundo constructor que recibe como parámetro un `NameComponent[]` y un tercero que recibe un String con el nombre del objeto en el formato devuelto por la función `toString()` de la propia clase `NombreMiCasmim`. Este último realiza la operación inversa de dicha función, generando a partir de un String un `NameComponent[]` y asignándolo a la estructura de datos *value* y tiene como pre-condición que el SEPARADOR no puede formar parte del String que se pasa como atributo ya que el mismo no es escapeado.

Para ver detalles relacionados a las operaciones implementadas referirse al Apéndice III: *Documentación de Capa de Red.*

9.7.2 MiCasmimDefs

Como parte de la especificación miCasmim realizada para nuestro proyecto se definió para cada tipo de objeto un String que lo identifica y una estructura jerárquica de nombre.

Para facilitar la implementación de los componentes se creo la clase `MiCasmimDefs.java` incluida en el paquete `mitiNum.nms.provisioning.miCasmimImpl`. La que incluye los tipos de objeto, tipos de estado de una connection, los tipos de errores y las funciones necesarias para construir el nombre de cada uno de los objetos miCasmim.

Estructura de datos

Tipos de objetos:

```
public static String TIPO_CONN="CONNECTION";
public static String TIPO_SN="SUBNETWORK";
public static String TIPO_TERM="TERMINATION";
```

Tipos de estado de una connection:

```
public static String ESTADO_EMPTY="EMPTY";
public static String ESTADO_IMPLEMENTED="IMPLEMENTED";
public static String ESTADO_RESERVED="RESERVED";
public static String ESTADO_OPERATIONAL="OPERATIONAL";
public static String ESTADO_REMOVED="REMOVED";
```

Constantes utilizadas para almacenar en el Inventario los parámetros (common features y termination features) de una connection:

```
public static String PAR_C_CF="PAR_C_CF";
public static String PAR_C_TF="PAR_C_TF";
public static String PAR_C_TID="PAR_C_TID";
```

Tipos de errores reportados por las interfaces miCasmim (Subnetwork2, Connection2 y Termination2):

```
public static String ERR_INVENTARIO="ERR_INVENTARIO";
public static String ERR_IMPL_CONN="ERR_IMPL_CONN";
public static String ERR_SISTEMA="ERR_SISTEMA";
```

Otras constantes utilizadas:

```
public static String SEP=".";
public static String ID="ID";
public static String IDL_VERSION="1.0";
```

Para ver detalles relacionados a las operaciones implementadas referirse al Apéndice III: *Documentación de Capa de Red.*

9.8 Routing

9.8.1 Análisis y diseño

El Routing es el componente de capa de red responsable de proveer los datos necesarios para crear las SNCs (y las rutas asociadas a ellas) de capa de elemento que componen un servicio de conectividad. Este componente es una fachada de granularidad gruesa que permite el acceso a los objetos MTNM mencionados (SNCCreateData_T).

Este componente es en un componente interno a la capa de red, usado por el componente Connection2, al momento de crear un servicio de conectividad. Este es un componente en su estructura,de manera de seguir respetando la idea conceptual seguida a lo largo del proyecto, pero no cuenta con la inteligencia de un verdadero modulo de routing. Su funcionamiento es estático y responde de acuerdo a la topología y es configurado por medio de un archivo de configuración que es consumido en el momento que se levanta el componente. Recalcamos que como quedó definido en el alcance del proyecto, no se intentó en ningún momento generar un componente inteligente, ya que su complejidad escapa al alcance.

Alcance

Para la interface del componente Routing se detallan sus operaciones:

Routing2.

Operaciones implementadas:

- getRoute

Inicialización

Al instanciar un componente Routing, el mismo recibe en su constructor una lista de parámetros necesarios para su correcta inicialización. La lista de parámetros es la siguiente:

- RutasFile: nombre del archivo XML desde donde el componente obtiene la definición de las rutas que va a manejar para satisfacer los pedidos para la topología actual. En el prototipo el nombre del archivo es rutas.xml, el mismo puede ser cambiado utilizando el archivo de configuración de capa de red.[69]

Persistencia

La definición de las rutas, como se dijo anteriormente, es almacenada en el archivo XML utilizado durante la inicialización. El esquema del mismo fue definido en el archivo rutasInitSchema.xsd, dicho esquema realiza una correspondencia entre la estructura de los objetos MTNM y una estructura XML. El formato del esquema es el siguiente:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<!-- Define la coleccion de rutas -->
<xsd:element name="rutas" type="RutaCollection"/>

<!-- Defino la coleccion de subnetwork Connections -->
<xsd:complexType name="RutaCollection">
    <xsd:sequence>
        <xsd:element name="ruta" type="Ruta" minOccurs="0" maxOccurs="unbounded"/>
```

69 Ver sección 8.1 Inicialización y configuración.

```
    </xsd:sequence>
</xsd:complexType>

<!-- Defino la subnetwork Connection -->
<xsd:complexType name="Ruta">
    <xsd:sequence>
        <xsd:element name="aendtp" type="NombreMTNMType" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="zendtp" type="NombreMTNMType" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="subnetworkconnection" type="SubnetworkConnectionType"
minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="SubnetworkConnectionType">
    <xsd:sequence>
        <xsd:element name="aendtp" type="NombreMTNMType" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="zendtp" type="NombreMTNMType" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="crossconnect" type="CrossConnectType" minOccurs="1"
maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="rate" type="xsd:string"/>
    <xsd:attribute name="direction" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="TPDataType">
    <xsd:sequence>
       <xsd:element name="nomendtp" type="NombreMTNMType" minOccurs="1" maxOccurs="1"/>
       <xsd:element name="ingrtraffdescname" type="NombreMTNMType" minOccurs="0"
maxOccurs="1"/>
       <xsd:element name="egrtraffdescname" type="NombreMTNMType" minOccurs="0"
maxOccurs="1"/>
       <xsd:element name="transmparam" type="LayeredParamType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="tpmappingmode" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="CrossConnectType">
    <xsd:sequence>
       <xsd:element name="aendtp" type="NombreMTNMType" minOccurs="1"
maxOccurs="unbounded"/>
       <xsd:element name="zendtp" type="NombreMTNMType" minOccurs="1"
maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="active" type="xsd:boolean"/>
    <xsd:attribute name="direction" type="xsd:string"/>
    <xsd:attribute name="cctype" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="LayeredParamType">
    <xsd:sequence>
       <xsd:element name="transmparamname" type="NombreMTNMType" minOccurs="1"
maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="layer" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="NombreMTNMType">
    <xsd:sequence>
       <xsd:element name="nom" type="NombreValor" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="NombreValor">
    <xsd:attribute name="nombre" type="xsd:string"/>
    <xsd:attribute name="valor" type="xsd:string"/>
```

```
</xsd:complexType>
</xsd:schema>
```

Para obtener información acerca de XML/XSD ver Apéndice II: *XML/XSD*.

Interface

La interface Routing2 esta definida en la idl que se muestra a continuación.

```
//
// mirouting.idl
//

#include "miconnection.idl"
#include "subnetworkConnection.idl"

module mitiNum {
module nms {
module provisioning {
module miCasmim {

typedef sequence<subnetworkConnection::SNCCreateData_T> SNCCreateData_TList;

    interface Routing2 {

              void getRoute(in FeatureInstances commonFeatures,
                        in Connection2::FeaturedTerminationContainers2 terminations,
                        out SNCCreateData_TList sncCDList )
                        raises (Common::Error::CommonException);
     };

};  // End of module 'miCasmim'
};  // provisioning
};  // nms
};  // mitiNum
```

9.8.2 Detalles de implementación

Estructura de datos

El comonente usa la clase CargarRutas para cargar una estructura que representa lo especificado en el archivo xml durante el proceso de inicialización, luego esta estructura se mantiene idéntica durante la vida del componente.

La estructura de datos que mapea el archivo xml leído es generada por JAXB [19] al momento de compilar el esquema, y por tanto esta formada por las clases del package rutasInit. No se optimizó ya que el alcance del componente no alienta una reutilización de ella, ya que en funcionamiento normal no existiría una estructura con estos fines, sino que por el contrario se usarían algoritmos dinámicos como backtraking y heurísticas para resolver los pedidos.

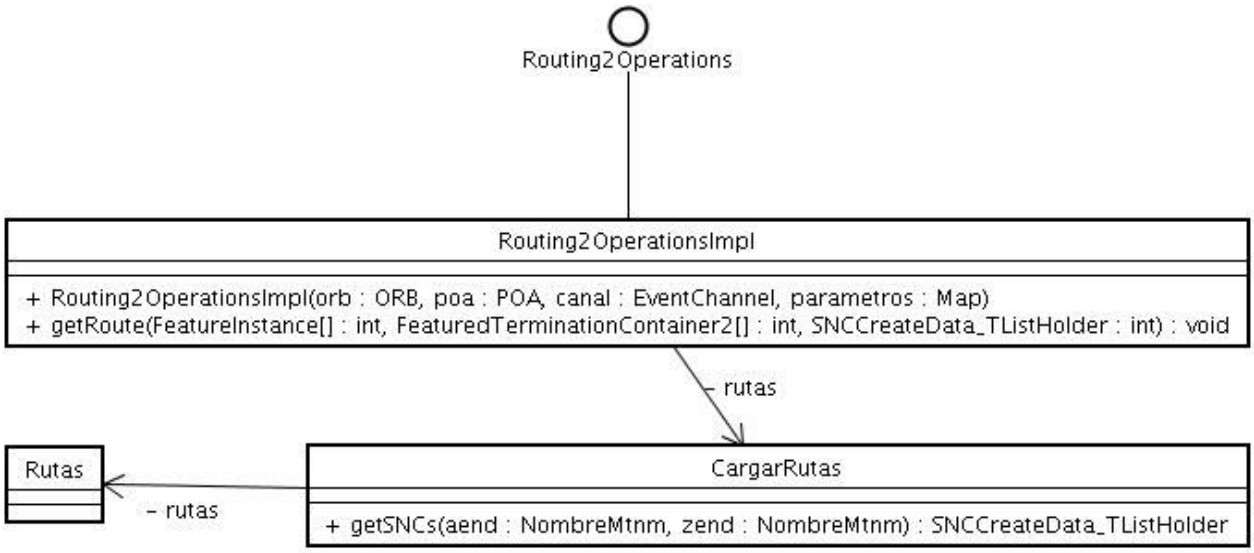


Diagrama 46 Routing

9.9 Cliente de capa de red: nmsGUI

Con el objetivo de probar la interface de miCasmim de Provisioning se creo una aplicación gráfica de nombre nmsGUI.

NmsGUI obtiene una referencia a las interfaces de Provisioning: Subnetwork2, Connection2 y Termination2 a través del servicio de nombres de CORBA.

NmsGUI permite al usuario realizar los siguientes casos de uso:

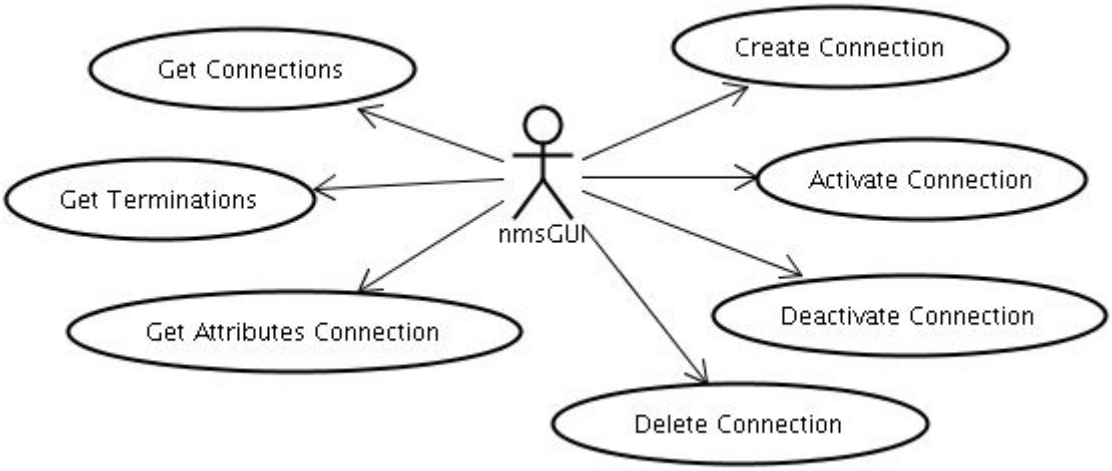


Diagrama 47 Diagrama de Casos de Uso

La aplicación tiene la particularidad que cada vez que actualiza sus vistas lo hace directamente contra Provisioning lo que nos permitió generar un tráfico considerable en CORBA para testear miCasmimm. En el mundo real la aplicación debería trabajar con algún tipo de cache.

Para conocer más sobre nmsGUI referirse al Apéndice IX: *Manual de usuario NMS GUI*.

10 Conclusiones

La industria de las telecomunicaciones está en pleno auge y constantemente se introducen cambios para proporcionar servicios de alta calidad. Debido a esto la gestión de las redes de telecomunicaciones evoluciona permanentemente para afrontar estos cambios intentando adelantarse a los mismos mediante el uso de estándares.

La tarea no es fácil, pero cada vez son mas las organizaciones que dedican recursos al estudio y al empleo de estándares. Algunas arquitecturas como la pirámide TMN y la separación en áreas funcionales (FCAPS) sirvieron como base para la gran mayoría de los estándares existentes en el mercado. Si bien las ideas iniciales persisten, las mismas fueron evolucionando incorporando paradigmas y herramientas de las tecnologías de la información (IT).

En el transcurso de nuestro proyecto se introdujeron cambios importantes a varios de los estándares investigados y utilizados en la implementación de nuestro sistema. Esto nos permite afirmar que el área continúa en constante evolución e integración. A modo de ejemplo queremos mencionar el cambio de versión de la interface MTNM y la mejora permanente de distribuciones de CORBA como JacORB. No podemos olvidar que la OMG participa activamente en la especificación de servicios CORBA que nacieron como necesidad de la industria de las telecomunicaciones, como es el caso del servicio de notificaciones de CORBA. El impacto de un estándar en el mercado está fuertemente condicionado por el peso de las organizaciones que lo promueven. Un ejemplo de esto es MTNM que cuenta con un fuerte apoyo de ATM. Esto hace que tenga muy en cuenta las necesidades de la industria, cosa que no siempre sucede con los proyectos puramente académicos.

Si bien pudimos cumplir con los objetivos que nos habíamos planteado al iniciar este proyecto no tenemos duda que queda mucho trabajo por hacer hasta contar con una solución completa de Gestión de Redes. Nuestro proyecto es un inicio que puede ser utilizado como punto de partida siempre que se tenga en claro la necesidad constante de evolución.

Con respecto a las tecnologías empleadas creemos que la elección de Java y CORBA permite que los componentes desarrollados se puedan integrar y reutilizar en el futuro, independientemente de la plataforma elegida. En particular, el uso de CORBA deja el camino abierto a otros lenguajes en partes del sistema en donde sea imposible el uso de Java, por ejemplo por razones de performance. Además ambas tecnologías son por naturaleza adecuadas para la programación orientada a componentes.

Las virtudes de la forma de trabajo orientada a componentes, empleando interfaces estándar y patrones de diseño son notorias en nuestro proyecto. Estas nos permitieron interactuar, sin demasiada gestión mediante, a dos grupos de trabajo que tenían en principio sus propios objetivos. Según nuestra experiencia no son necesarias gran cantidad de reuniones intergrupales y por lo tanto esta metodología de trabajo se puede proponer para grupos que se encuentren físicamente en lugares distantes.

Uno de los resultado más interesantes de este proyecto es que deja varios caminos abiertos para futuros trabajos. Algunos de los temas que han sido identificados durante el desarrollo del proyecto fueron:

- Ruteo
  El componente de ruteo desarrollado para el proyecto es ficticio, ya que las rutas devueltas son pre-cargadas en un archivo de configuración. Un componente de ruteo es imprescindible para una solución real, y ofrece muchas vías de investigación. Este es un área en constante evolución, conocida como “Ingeniería de Tráfico”.
- Componente de aprovisionamiento.
  Como se indicó, el componente de aprovisionamiento utiliza una versión simplificada de una interface estándar (CaSMIM). Sería interesante levantar esas restricciones, y posiblemente estudiar un integración más completa con el modelo WINMAN (utilizado por el Inventario de Red desarrollado por el proyecto Gesinv)
- Persistencia en capa de elemento
  Si bien este proyecto muestra que es posible que cada fachada MTNM maneje su propia persistencia independientemente de las otras, sería interesante investigar las ventajas que se podrían obtener al utilizar una capa integradora, de forma de unificar el manejo de la persistencia.
- Manejo de transacciones
  Sería interesante estudiar si se podría mejorar MTNM al introducir el concepto de transacción. Buena

parte de la complejidad del área de aprovisionamiento de capa de elemento proviene de la gestión de anomalías durante operaciones complejas, que afectan la configuración de multitud de dispositivos. Esta complejidad se manifiesta en las interfaces en la forma de estados, que deben ser manejados explícitamente. En la práctica, los componentes y clientes terminan resolviendo problemas clásicos de la gestión de Bases de Datos, como son la concurrencia y la indivisibilidad.
Este mismo análisis se podría repetir para la Capa de Red.

- Aplicación a una tecnología de redes real
  Este es un paso imprescindible si se quiere evolucionar hacia una solución práctica para el problema de la Gestión de Redes. El sistema tal como está soporta una clase de dispositivos ficticia, con capacidades inventadas. Seguramente al intentar agregar soporte para una tecnología real surjan omisiones y falencias que se cometieron durante el desarrollo. Un buen candidato para empezar es MPLS, por ser una solución sobre IP para servicios orientado a conexión con calidad de servicio, y tener interés académico.
- Re-ingeniería de la arquitectura
  Durante el desarrollo se tuvo muy en cuenta el criterio de la simplicidad y la claridad, en desmedro de la escalabilidad o la elegancia general de la solución. Algunos elementos de la arquitectura se verían favorecidos con un segundo análisis. Por ejemplo, los iteradores de MTNM podrían ser mejor encapsulados.
- Seguridad
  El tema de la seguridad y gestión de sesiones necesita ser resuelto. Este proyecto únicamente identificó las dificultades y desarrolló un esquema en donde una solución real podría inscribirse.
- Alinearse con MTNM
  En la actualidad hay sistemas que no están completos, como por ejemplo la gestión de filtros para las notificaciones. Hay componentes completos que faltan, como por ejemplo la gestión de equipamiento. También hay que tener en cuenta que la especificación MTNM siguió evolucionando, por lo que sería bueno evaluar los cambios introducidos, en particular en la versión 3.0.
- Configuración
  Para que una solución sea aplicable en un entorno real, necesita un marco robusto para la configuración, inicialización y monitoreo de los componentes. Este marco debería ofrecer configuración centralizada, alta disponibilidad, resistencia a fallos, flexibilidad para gestionar instalaciones distribuidas, etc.

11 Apéndice I: Documentación de Capa de Elemento

El Apéndice I incluye la documentación complementaria relacionada con la capa de elemento no incluida en la sección 8. *Capa de elemento* de éste documento. El contenido del apéndice puede ser encontrado en el documento *anexo de apéndices*[70]. La información incluida es la siguiente:

11.1 NombreMtnm

Descripciones de cada una de las operaciones públicas en formato tabular, incluyendo detalles de implementación.

11.2 MTNMDefs

Descripciones de cada una de las operaciones públicas en formato tabular, incluyendo detalles de implementación.

11.3 EMS Manager

Descripciones de cada una de las operaciones públicas y privadas en formato tabular, incluyendo detalles de implementación.

11.4 MLSN Manager

Descripciones de cada una de las operaciones públicas y privadas en formato tabular, incluyendo detalles de implementación.

11.5 Alcance de EmsGui

Documentación complementaria relacionada con la aplicación cliente de capa de elemento no incluida en la sección 8.9 *Cliente de capa de elemento: emsGUI*.

70 Los apéndices de éste documento fueron incluidos todos juntos en un documento *anexo de apéndices* por razones de orden y comodidad para el lector.

12 APÉNDICE II: XML/XSD

El Apéndice II incluye detalles relacionados a los estándar XML/XSD y la forma en que fueron utilizados para el manejo de los archivos de inicialización y persistencia de los componentes de **mitiNum**. El contenido del apéndice puede ser encontrado en el documento *anexo de apéndices*[71]. La información incluida es la siguiente:

12.1 XML/XSD

Descripción de los motivos relacionados al uso de los estándar XML/XSD y de la herramienta JAXB [19] utilizada como parser.

71 Los apéndices de éste documento fueron incluidos todos juntos en un documento *anexo de apéndices* por razones de orden y comodidad para el lector.

13 Apéndice III: Documentación de Capa de Red

El Apéndice III incluye la documentación complementaria relacionada con la capa de red no incluida en la sección 9. *Capa de red* de éste documento. El contenido del apéndice puede ser encontrado en el documento *anexo de apéndices*[72]. La información incluida es la siguiente:

13.1 Términos clave

Definiciones de los términos relacionados con capa de red: features, connections, entro otros.

13.2 CaSMIM y TOM

Procesos de desarrollo del TOM incluidos en CaSMIM y cuales de ellos fueron incluidos y no incluidos en miCasmim. Esta información complementa la definición del alcance de miCasmim.

13.3 Requerimientos

Requerimientos que cubre CaSMIM y cuales de ellos fueron cubiertos y cuales no en miCasmim. Esta información complementa la definición del alcance de miCasmim.

13.4 Casos de Uso

Casos de uso relacionados a los requerimientos de CaSMIM cubiertos por miCamim según el alcance de nuestro proyecto.

13.5 NombreMiCasmim

Descripciones de cada una de las operaciones públicas en formato tabular, incluyendo detalles de implementación.

13.6 MiCasmimDefs

Descripciones de cada una de las operaciones públicas en formato tabular, incluyendo detalles de implementación.

13.7 Subnetwork2

Descripciones de cada una de las operaciones públicas y privadas en formato tabular, incluyendo detalles de implementación.

13.8 Connection2

Descripciones de cada una de las operaciones públicas y privadas en formato tabular, incluyendo detalles de implementación.

13.9 Termination2

Descripciones de cada una de las operaciones públicas y privadas en formato tabular, incluyendo detalles de implementación.

13.10 Routing2

Descripciones de cada una de las operaciones públicas y privadas en formato tabular, incluyendo detalles de implementación.

---

72 Los apéndices de éste documento fueron incluidos todos juntos en un documento *anexo de apéndices* por razones de orden y comodidad para el lector.

14 APÉNDICE IV: IDLS MICASMIM

El Apéndice IV incluye las IDLs realizadas para ser usadas como interfaces del componente Provisioning de capa de red de **mitiNum**. El contenido del apéndice puede ser encontrado en el documento *anexo de apéndices*[73]. La información incluida es la siguiente:

14.1 IDLs miCasmim

Se incluyen las IDLs micommon.idl y miconnection.idl de miCasmim, las mismas se realizaron tomando como partida sus pares en CaSMIM y se indica cuales fueron los cambios o modificaciones según el alcance de nuestro proyecto.

73 Los apéndices de éste documento fueron incluidos todos juntos en un documento *anexo de apéndices* por razones de orden y comodidad para el lector.

15 Apéndice V: Instructivo de Deployment

El Apéndice V incluye los requerimientos previos y los pasos necesario para realizar la instalación de **mitiNum** a partir de su CD de instalación. El contenido del apéndice puede ser encontrado en el documento *anexo de apéndices*[74].

74 Los apéndices de éste documento fueron incluidos todos juntos en un documento *anexo de apéndices* por razones de orden y comodidad para el lector.

16 APÉNDICE VI: PRUEBAS

El Apéndice VI incluye un resumen de las pruebas realizadas sobre el sistema. El contenido del apéndice puede ser encontrado en el documento *anexo de apéndices*[75]. La información incluida es la siguiente:

16.1 Configuración de pruebas

Configuración de pruebas y resultados obtenidos relacionados con el consumo de recursos.

16.2 Instalación distribuida

Como instalar el sistema en forma distribuida.

16.3 Escalabilidad

Posibilidades de escalabilidad del sistema.

16.4 Topología de prueba

Diagramas de la subnetwork SubredTest utilizada para probar el sistema.

75 Los apéndices de éste documento fueron incluidos todos juntos en un documento *anexo de apéndices* por razones de orden y comodidad para el lector.

17 APÉNDICE VII: MANUAL DE USUARIO EMS CONFIG GUI

EMS Config GUI es una aplicación gráfica de configuración de capa de elemento. Permite configurar la topología de la red a demanda. Un usuario podrá en cualquier momento crear, modificar o eliminar MultiLayerSubnetworks y TopologicalLinks.

El contenido del apéndice puede ser encontrado en el documento *anexo de apéndices*[76].

76 Los apéndices de éste documento fueron incluidos todos juntos en un documento *anexo de apéndices* por razones de orden y comodidad para el lector.

18 APÉNDICE VIII: MANUAL DE USUARIO EMS GUI

EMS GUI es una aplicación gráfica que permite navegar los objetos de capa de elemento. Un usuario podrá en cualquier momento visualizar MultiLayerSubnetworks, TopologicalLinks, SubnetworkConnections y ManagedElements en una vista de árbol. Además podrá modificar algunas de las propiedades de los objetos y crear, activar, desactivar o eliminar SubnetworkConnections.

El contenido del apéndice puede ser encontrado en el documento *anexo de apéndices*[77].

77 Los apéndices de éste documento fueron incluidos todos juntos en un documento *anexo de apéndices* por razones de orden y comodidad para el lector.

19 APÉNDICE IX: MANUAL DE USUARIO NMS GUI

NMS GUI es una aplicación gráfica que permite navegar los objetos de capa de red. Un usuario podrá en cualquier momento crear, modificar o eliminar Connections.

Es una aplicación de capa de red y no de capa de servicio y por lo tanto no está pensada para un usuario de nivel gerencial. Un servicio de conectividad podría implicar la creación de una o mas connections. En ésta interface se trabaja en términos de connections y no de servicios de conectividad.

El contenido del apéndice puede ser encontrado en el documento *anexo de apéndices*[78].

78 Los apéndices de éste documento fueron incluidos todos juntos en un documento *anexo de apéndices* por razones de orden y comodidad para el lector.

20 GLOSARIO

ADSL	Asymmetrical Digital Subscriber Line
ASN.1	Abstract Syntax Notation One Protocolo estándar de presentación de ISO utilizado por SNMP para representar mensajes.
ATM	Asynchronous Transfer Mode.
BMS	Business Management System
CaSMIM	Connection and Service Management Information Model Estandar del TeleManagement Forum para gestión de servicios de conectividad.
CDP	Cisco Discovery Protocol.
CMIP	Common Management Information Protocol
CMIS	Common Management Information Service
CORBA	Common Object Request Broker Architecture
CTP	**C**onnection **T**ermination **P**oint Un CTP representa el actual o potencial terminación de conexión de una SubnetworkConnection (definición según la especificación MTNM).
CVS	Concurrent Versions System
DLC	Data Link Connection. Un DLC representa un segmento de un circuito virtual entre un dispositivo final de usuario y un dispositivo frame relay, o entre dispositivos de red frame relay.
DLCI	Data Link Conection Identifier. Un DLCI identifica a un DLC. Los DLCIs solo necesitan ser únicos en en un cierto frame stream; no en toda la red. Todos los frames que tienen el mismo DLCI y que son llevados dentro del mismo stream frame son asociados con su conexión lógica.
DS0	Refiere a la señal digital de nivel cero: 64Kb/s. Este es un estándar mundial para voz digitalizada PCM (Pulse Code Modulation).
EMS	Element Management System
FCAPS	Areas funcionales de los sistemas de gestión de las redes de telecomunicaciones. **F**ault, **C**onfiguration, **A**ccounting, **P**erformance, **S**ecurity.

Frame Relay	Es un protocolo de switcheo de paquetes similar a X.25, sin embargo, requiere menos procesamiento y está diseñado para operar a velocidades mucho mayores. Muchas terminales remotas de LAN, pueden usarlo para compartir ancho de banda de un DS0 en un link T1.
GDMO	Guidelines for the Definition of Managed Objects
HFC	Hybrid fibre-coax Las llamadas redes de acceso.
JMAPI	Java Management Application Programming Interface
LMI	Un protocolo usado para el descubrimiento de ATM
IDL	CORBA Interface Definition Language
IOR	Interoperable Object Reference
ISDN	ISDN, es un estándar para **I**ntegrated **S**ervices **D**igital **N**etwork, es un sistema de conexiones telefónicas que han estado disponibles desde décadas. Este sistema permite transmisión simultanea de voz y de datos a través del mundo usando conexiones digitales end-to-end.
ISP	Internet Service Provider
IT	Information Technology
ITU-T	International Telecommunication Union-Telecommunication Standardization
JAXB	*Java XML Binding*
JIDM	Joint Inter-Domain Management
LAN	Local Area Network.Es una red que opera dentro de un area delimitada geograficamente, como en un edificio. Conecta una variedad de dispositivos de datos, como PCs, servidores e impresoras. La comunicación entre dispositivos es a una gran velocidad de datos.
MAC	Media Access Control.La capa MAC es una subcapa de la capa de enlace del stack OSI. Está definida por el IEEE 802.2. Toda computadora y nodo de la red tiene su direccion MAC que esta codificada en el hardware.
ME	Los **M**anaged **E**lement representan a los dispositivos gestionados según la especificación MTNM.
MIB	Management Information Base. La MIB es la base de datos de un dispositivo administrado por SNMP.
MoIP	Multimedia over IP

MPLS	Multiprotocol Label Switching. Es una tecnica de switcheado orientada a la coneccion, basada en routeo IP y protocolos de control.
MTNM	Multi-Technology Network Management
MLSN	**M**ulti**L**ayer**S**ub**n**etwork o Subnetwork es una abstracción provista por el EMS al el NMS. La Subnetwork es la unidad de topología de un sistema EMS (definición según la especificación MTNM).
NE	Network Element
NMS	Network Management System
ORB	Object Request Broker
OSS	Operations Support System.
OSI	Open Systems Interconnection
POA	Portable Object Adapter
PTP	**P**onnection **T**ermination **P**oint Un PTP representa el actual o potencial terminación de conexión de un Topological Link, esencialmente es la representación de un puerto físico (definición según la especificación MTNM).
PVC	Private Virtual Circuits.
QoS	Quality of Service.
RDBMS	Relational Database Management Systems
RMI	Remote Method Invocation
RMON	Usado para el monitoreo de trafico.
RMON2	Usado para el monitoreo de trafico.
root interfaces	Según la especificación MTNM la fachada EmsSessionFactory junto con los componentes EMSSession y NMSSession componen lo que se conoce como *root interface*. Este subsistema permite acceso a los recursos de un sistema EMS.
Router	Un router es un dispositivo de interconexion LAN/LAN o LAN/WAN. Selecciona la ruta de costo mas efectivo para el flujo de datos entre LANs de muchos protocolos, asegurando que solo exista una ruta entre los dispositivos fuente y destino.
SDH	Synchronous Digital Hierarchy.

SMS	Service Management System
SNC	Una **S**ub**n**etwork **C**onnection relaciona terminaciones de conexión (CTPs) y ofrece una conexión end-to-end trasparente a través de una subnetwork (según la especificación MTNM).
SNMP	Simple Network Management Protocol. Es el protocolo estándar para la administracion de LANs. Originado en la comunidad Internet como respuesta a la nececidad de administrar dispositivos TCP/IP. SNMP da aceso a la MIB del dispositivo.
SONET	Synchronous Optical Network.
SSL	Secure Socket Layer
TDM	Time Divition Multiplexing.
TINA	Estandar del consorcio TINA-C (Telecommunications Information Networking Architecture Consortium).
TL	**T**opological **L**ink representan los enlaces físicos entre elementos gestionados (MEs), mas precisamente entre dos PTPs o en link ATM entre dos CTPs, según la especificación MTNM.
TL1	Transaction Language One Es un protocolo de gestión en telecomunicaciones.
TMN	Telecommunication Management Network
TMF	TeleManagement Forum
T1	Es un estándar de transmision digital a 1544 Mb/s. Es el estándar generalmente usado en Norte America y Japon. Tambien es conocido como DS1. El equivalente T1 para Europa es el E1.
TOM	Telecom Operations Map
TP	**T**ermination **P**oint Un TP representa el actual o potencial terminación de conexión de una SubnetworkConnection o un Topological Link o de un grupo de TPs y están contenidos en un Managed Element (definición según la especificación MTNM). Un TP puede ser de alguno de los siguientes tipos: TP físico (PTP), TP de conexión (CTP) o grupo de TPs (TPPool).
UML	Unified Modeling Language
VoIP	Voice over IP
VPNs	Virtual Private Networks

WAN	Wide Area Network. Una WAN es una red de conexion de transmision digital.
WDM	Wavelength Division Multiplexing
WINMAN	WDM and IP Network MANagement
X.25	Es un protocolo basado en paquetes que provee confiabilidad y garantía de entrega de datos end-to-end.
XML	eXtensible Markup Language
XSD	*XML Schema Definition*

21 REFERENCIAS

[1] Operations Support System (OSS), www.iec.org

[2] Telecommunications Management Network (TMN), www.onforum.com/tutorials/

[3] TMN & IN, www.doc.ic.ac.uk/~nd/surprise_97/journal/vol4/tkl1

[4] TMN Evolution, www.eurescom.de

[5] ITU-T, www.itu.int/ITU-T/

[6] TMForum, www.tmforum.com

[7] SNMP, www.cisco.com/univercd/cc/td/doc/product/webscale/css/advcfggd/snmp.htm

[8] TINA,www.ee.surrey.ac.uk/Personal/G.Pavlou/Publications/Conference-papers/Ranc-97.pdf

[9] Java, www.sun.com/java

[10] OMG, www.omg.org

[11] CORBA, www.onforum.com/tutorials/11Weikopf.ppt

[12] HPOV, www.docs.hp.com

[13] CISCO, www.cisco.com/univercd/cc/td/doc/

[14] Alcatel 5620, www.alcatel.com

[15] Adventnet, www.adventnet.com

[16] XML, http://www.w3.org/XML/

[17] WINMAN Web Site: http://www.winman.org

[18] JacORB, www.jacorb.org

[19] JAXB, www.sun.com

[20] Postgres, www.postgresql.org