Universidade de Lisboa
Faculdade de Ciências
Departamento de Informática

Suporte à Gestão Topológica de Redes de Telecomunicações

Ana Margarida Martins Ferreira

Mestrado em Engenharia Informática

2007

Universidade de Lisboa
Faculdade de Ciências
Departamento de Informática

Suporte à Gestão Topológica de Redes de Telecomunicações

Ana Margarida Martins Ferreira

Projecto Orientado pelo Prof. José Manuel do Sousa de Matos Rufino
e co-orientado pelo Mestre João Filipe Robalo Neves Leal

Mestrado em Engenharia Informática

2007

Resumo

Assistimos nos últimos anos a um verdadeiro “boom” na área das telecomunicações. A forma e a facilidade como comunicamos hoje em dia é completamente diferente face à realidade de alguns anos atrás. Os serviços são melhores, mais rápidos, mais cómodos. O mundo das telecomunicações está em constante mudança devido a exigência do mercado. Mais disponibilidade de serviço, maior largura de banda, mais e melhores funcionalidades para os produtos são algumas dos objectivos das operadoras. O âmbito do projecto aqui apresentado prende-se com um sistema baseado numa arquitectura de *software* distribuída e escalável aplicada à gestão de redes de telecomunicações. Este sistema permite às operadoras a gestão da sua rede de uma forma rápida, simples e eficaz.

Palavras - Chave:

- Telecomunicações
- Software de Gestão de Redes
- J2EE
- Arquitectura Modular
- Sistemas Distribuídos

Abstract

The world of telecommunications it's always changing. The way we communicate these days is completely different than a few years ago. People are demanding for much faster services, more bandwidth, and greater availability. The changing requirements and the rising amount of new services claimed by consumers and the increasing cost pressures are challenges service providers have to take up. To meet with these challenges it is essential to develop an adequate and flexible network concept based on appropriate and innovative network elements. The market commands the evolution in this area. In this document it is presented a modular, flexible and reliable architecture for distributed telecommunications network management software. This system allows the operators to manage a whole network from one single place in a simple, easy, clever and fast way.

Keywords:

- Telecommunications
- Network Management Software
- J2EE
- Modular Arquitecture
- Distributed Systems

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Assistimos nos últimos anos a uma verdadeira proliferação de serviços na área das telecomunicações. A forma e a facilidade como comunicamos hoje em dia é completamente diferente face à realidade de alguns anos atrás. Os serviços são melhores, mais rápidos, mais cómodos. O mundo das telecomunicações está em constante mudança devido à exigência do mercado. Cada vez mais, a maior disponibilidade de serviço, maior largura de banda, mais e melhores funcionalidades para os produtos são os objectivos das operadoras.

Este projecto enquadra-se no desenvolvimento de dois produtos: O Telecommunication Network Management System Core (TNMS-CORE) e o Telecommunication Network Management System Domain Unix (TNMS-DX). Ambos os sistemas são baseados numa arquitectura de programação distribuída e escalável que aplicada à gestão de redes de telecomunicações permitem às operadoras a gestão da sua rede de uma forma rápida, simples e eficaz.

Estes produtos são duas soluções de gestão da Siemens S.A., para satisfazer as necessidades das redes de telecomunicações metropolitanas de transporte de dados bem como das redes de telecomunicações internacionais ou nacionais de longa distância.

O TNMS-DX é um produto que já se encontra em desenvolvimento há cerca de dois anos. A equipa de desenvolvimento deste projecto tem perto de cinquenta pessoas espalhadas por três países: Portugal, Alemanha e Índia. Correntemente este produto encontra-se na versão três ponto zero.

Outro produto que já se encontra há algum tempo na fase de desenvolvimento é o TNMS-CORE. A sua equipa de desenvolvimento conta com cerca de cem pessoas espalhadas por quatro países: Portugal, Alemanha, Índia e China. Actualmente este produto encontra-se na versão onze, onde a sua arquitectura sofreu uma reformulação substancial.

Estes produtos oferecem todos os elementos e serviços para desenvolver, construir e gerir redes de telecomunicações. Destes produtos fazem parte componentes de *Hardware*, equipamentos físicos que suportam a infra-estrutura física da rede e componentes de *Software* que permitem a gestão da mesma.

Ambos os produtos podem ser dimensionados e adaptados consoante as diferentes necessidades de soluções para gestão de redes. São sistemas de gestão multi-utilizador com uma arquitectura do tipo cliente-servidor que oferecem a possibilidade de visualizar a estrutura de uma rede inteira de telecomunicações e de a gerir. Os dois sistemas fornecem uma interface gráfica que pode ser acedida através de um cliente que permite o acesso a todas as funções do sistema: gestão de serviços, gestão da rede, gestão dos elementos de rede (NE do inglês Network Elements) que são dispositivos físicos de telecomunicações.

A arquitectura destes sistemas é bastante complexa dividindo-se em vários componentes. Existem no entanto componentes genéricos que podem ser reutilizados transversalmente em vários produtos de gestão, denominados Funções Comuns (CF do Inglês Common Functions). Estes componentes devido a seu carácter modular fazem parte da arquitectura de ambos os produtos. O trabalho realizado durante o estágio foi desenvolvido no contexto de uma Função Comum que suporta todas as funcionalidades necessárias para a definição e visualização dos dados relativos à descrição da rede física – Gestão Topológica (do Inglês Topology Management).

1.1 Apresentação da Empresa de Acolhimento

O projecto aqui apresentado foi realizado na Siemens S.A. uma das maiores companhias de produtos electrónicos do mundo. Fundada em 1847 em Munique, na Alemanha, onde actualmente continua sediada, esta é uma empresa de prestígio reconhecido internacionalmente. Está presente em vários mercados sendo que as principais áreas de negócio são: Informação e Comunicação, Energia, Automatização e Controlo, Transporte e Medicina. No início actuava essencialmente no fabrico de equipamentos de telecomunicações mas actualmente está também nas áreas de material eléctrico, infra-estrutura do sector energético (eléctrico e nuclear), transporte público (na construção de comboios e metros), equipamento hospitalar e computadores (Fujitsu-Siemens). Em todo o mundo a Siemens emprega cerca de 461.000 pessoas em mais de 190 países.

A empresa comemorou no ano de 2005 o centenário no nosso país onde possui instalações nas zonas de Alfragide, Porto, Coimbra e Aveiro. Apesar das áreas de maior foco serem sem dúvida as da indústria eléctrica e electrónica possui também departamentos de Investigação e Desenvolvimento (do Inglês Research And Development - R&D) que participam activamente na rede mundial de inovação.

Nas instalações de Alfragide existem nos dois departamentos de R&D e três laboratórios (multimédia, redes ópticas e gestão de redes de telecomunicações) cuja qualidade e excelência é reconhecida mundialmente. O trabalho realizado nestas instalações enquadra-se na área de negócio Informação e Comunicação. Actualmente esta área detém a maior parte das vendas da empresa e está em constante expansão, justificando-se desta forma o forte investimento e a aposta da Siemens em território Luso.

Recentemente e com o objectivo mútuo de se tornarem número um a nível mundial no fornecimento de equipamento e sistemas de telecomunicações para redes fixas e móveis, foi realizada uma fusão entre a Nokia Networks e a Siemens Networks. No dia 1 de Abril de 2007 foi anunciada a consolidação da fusão e o nascimento de uma nova empresa, a Nokia-Siemens Networks. Devido ás especificações da fusão, os departamentos cuja área de negócio é Informação e Comunicação foram transferidos da Siemens Networks para a Nokia-Siemens Networks.

1.2 Apresentação da Equipa

Nas instalações da Siemens Alfragide estão sediados os departamentos de Investigação e Desenvolvimento: R&D1 e R&D2. O TNMS-CORE e o TNMS-DX são ambos produtos de gestão de redes de telecomunicações e estão a ser desenvolvidos no R&D1. A sua arquitectura é bastante complexa e como tal encontra-se dividida em cerca de vinte e dois componentes. Cada componente está afecto a uma equipa e cada uma tem um coordenador que se denomina de *Team Leader*. Este é responsável pela concretização dos objectivos propostos à mesma.

A equipa responsável pelas Funções Comuns, tem como líder de equipa o Mestre João Leal e no total conta com cerca de treze colaboradores. Esta equipa divide-se em várias sub-equipas, cada uma afecta a uma Função Comum específica.

1.3 Enquadramento e Objectivos

O trabalho realizado encontra-se enquadrado no desenvolvimento de dois produtos de gestão de redes de telecomunicações, duas soluções de gestão para satisfazer as necessidades das redes de telecomunicações metropolitanas de transporte de dados bem como das redes de telecomunicações internacionais ou nacionais de longa distância.

O grande objectivo deste projecto é o estudo e concretização de uma arquitectura de *Software* distribuída e escalável, com vista ao suporte de aplicações de gestão e controlo de redes de telecomunicações na área de transporte de dados. Especificamente todo o trabalho de engenharia centrou-se no desenvolvimento, optimização e manutenção de um componente responsável pela gestão, visualização e definição da topologia de redes de telecomunicações.

Durante o decorrer do projecto novo conhecimento é apreendido:

- Tirar partido das facilidades disponibilizadas por um servidor de aplicações para diminuir o tempo de desenvolvimento e aumentar a reutilização do componente.
- Explorar as capacidades de mapeamento objecto-relacional de um sistema como Hibernate para facilitar e optimizar o acesso a repositórios de informação.

Outro objectivo de natureza não técnica mas igualmente importante é a integração e a percepção da dinâmica de uma equipa de desenvolvimento.

1.4 Metodologia e Calendarização

Neste ponto é descrita a metodologia utilizada no desenvolvimento dos projectos na Siemens S.A e também a calendarização definida para a realização dos trabalhos aqui descritos.

1.4.1 Metodologia

O processo de desenvolvimento dos projectos na Siemens S.A segue uma linha e metodologia própria que foi sendo aperfeiçoada ao longo dos anos. Esta metodologia é aplicada a todos os produtos de grande dimensão de forma a assegurar a qualidade dos

processos e do produto. Como o tempo de vida de alguns projectos é bastante elevado é possível que exista integração de novos membros numa equipa em fases mais avançadas. Como tal é importante perceber as etapas e os objectivos a atingir em cada uma das fases deste processo.

O modelo de desenvolvimento dos projectos na Siemens S.A tem como base um modelo cascata ou clássico também conhecido como a abordagem *Top-Down,* mais detalhado e adaptado às necessidades da empresa. Comparado com outros modelos de desenvolvimento de *software* este é mais rígido e menos administrativo. Grande parte do sucesso do modelo cascata está no facto de ser orientado para documentação. No modelo cascata tradicional existem cinco fases: análise e definição de requisitos, desenho do sistema, implementação, testes e manutenção.

Na aproximação seguida pela Siemens S.A, o ciclo de vida de um processo consiste em identificar uma oportunidade de negócio e avalia-la de forma a verificar os requisitos e sua exequibilidade. Posteriormente é necessário traçar um plano de negócio para o produto e iniciar o lançamento do produto. O PPP (Product Provisioning Process) é o processo seguido no desenvolvimento do produto. Este encontra-se divido em vários subprocessos, no entanto é na fase PPP:Development (1), a de implementação do produto, que o trabalho realizado está enquadrado. Esta fase está dividida em várias sub-etapas que são atingidas mediante a concretização de objectivos pré-definidos. Este processo assegura a qualidade do produto e a sua estabilidade devido ao processo pelo qual este tem de ser submetido até ser libertado para o utilizador final ou seja o cliente.

O PPP:Development consiste em seis fases: Análise, Desenho, Implementação, Integração, Testes de Sistema (realizados por quem produz o produto) e Testes de Aceitação (realizados por quem compra o produto). As semelhanças com o modelo cascata tradicional são evidentes.

Algumas das fases estão divididas em sub-etapas como acontece na fase Implementação. Durante a fase de implementação existe um processo a seguir. Primeiro esboça-se o desenho do *software* e implementa-se uma primeira versão. Essa primeira versão é revista e melhorada e caso necessário é implementada uma segunda e definitiva versão do *software*. Após a implementação são realizados testes pela equipa que produziu o *software*. Este é o processo que é seguido aquando da realização de qualquer *software* produzido para integrar um produto ou sistema.

O trabalho realizado foi desenvolvido no contexto de dois produtos. Ambos os produtos são versões novas de produtos já existentes com a adição de novas

funcionalidades. Devido ao facto de serem considerados produtos novos estes tiveram de passar por todo o processo desde a entrega de documentação para a nova versão com a actualização das novas funcionalidades, o redesenho da arquitectura do sistema (quando necessário), o desenvolvimento das novas funcionalidades para cada um dos componentes e a concretização da nova arquitectura, a integração de todos os componentes do sistema e os testes.

1.4.2 Calendarização

O trabalho realizado no componente responsável pela Gestão Topológica da rede foi a implementação de novas funcionalidades ou optimização de algumas já existentes. Como tal foi necessário haver um período inicial de integração na equipa, no projecto e a leitura de documentação do componente onde seriam realizados os trabalhos. Este período é útil para conhecer o sistema onde se vai desenvolver o trabalho. Só após esta fase é possível o início da implementação das funcionalidades necessárias ao sistema e posterior teste das mesmas. Estes testes são testes unitários feitos pela equipa/pessoa que desenvolve a funcionalidade. Concluída esta fase há a integração de todos os componentes e testes realizados por uma equipa especializada que realiza testes mais elaborados de acordo com a especificação fornecida. Durante esta fase de testes podem surgir problemas pelo que se inicia uma fase de manutenção à funcionalidade. De um modo geral o planeamento seguido foi o seguinte:

1.Período: 09/2006
Descrição: Integração na equipa, leitura da documentação do projecto, integração no projecto.

2.Período:10/2006 – 12/2006
Descrição: Implementação de funções necessárias ao sistema.

3.Período: 01/2006 – 02/2006
Descrição: Implementação de testes unitários sobre as funções desenvolvidas.

4.Período: 03/2006
Descrição: Participação nas actividades de integração dos vários componentes.

5.Período: 04/2006 – 05/2006
Descrição: Participação nas actividades de correcção de erros e manutenção das funções.

6.Período: 06/2006
Descrição: Elaboração do relatório que descreve o trabalho efectuado nos períodos anteriores.

Desvios ao Plano Inicial

Em relação ao plano inicial houve um desvio devido a não contemplação no plano da elaboração do relatório que iria descrever o trabalho efectuado.

Para além do desvio referido a cima não houve mais nenhum desvio ao plano inicial.

1.4.3 Organização do Documento

O presente documento encontra-se organizado em quatro capítulos:

1. Introdução: Capítulo onde se pretende dar uma visão global do projecto e do enquadramento do mesmo. É também o capítulo onde estão definidos os objectivos delineados no início, a metodologia de desenvolvimento seguida e a calendarização do trabalho realizado.
2. Arquitecturas de Sistemas de Gestão de Redes: Neste capítulo é apresentado e descrito o modelo seguido pela Siemens S.A. para a concretização de sistemas de gestão de redes.
3. Gestão da Topologia da Rede: Neste capítulo é descrita de forma pormenorizada o componente responsável pela gestão topológica da rede e o seu enquadramento no Sistema de Gestão de Redes apresentado no Capítulo 2.
4. Conclusões: Este capítulo resume as principais conclusões recolhidas no decorrer no projecto.

Capítulo 2

Arquitecturas de Sistemas de Gestão de Redes

O trabalho apresentado foi realizado no contexto de um componente que suporta todas as funcionalidades de definição, gestão e visualização dos dados relativos a descrição de uma rede física. Este componente está integrado em sistemas de gestão de redes de telecomunicações. A arquitectura deste tipo de sistemas é muito complexa dividindo-se em vários componentes cada um com uma função específica. O tipo de arquitectura dos sistemas onde o componente responsável pela gestão da topologia da rede está integrado segue o modelo de uma arquitectura tridimensional definida no modelo da *Sun Architectural Methodology* (2). Este tipo de arquitectura é apresentado de seguida no documento.

2.1. Modelo: *Sun Architectural Methodology*

O *Sun Architectural Methodology* define um modelo que organiza a análise do *software* de um sistema em três dimensões: *tiers, layers* e *systemic qualities*. O modelo é graficamente representado como um cubo. A razão de ser um cubo é devido a existência de uma relação ortogonal entre *tiers, layers* e *systemic qualities*. Cada *tier* é suportada por um ou mais *layers* e as *systemic qualities* tem de ser persistentes na arquitectura tanto para as *tiers* como para as *layers*.

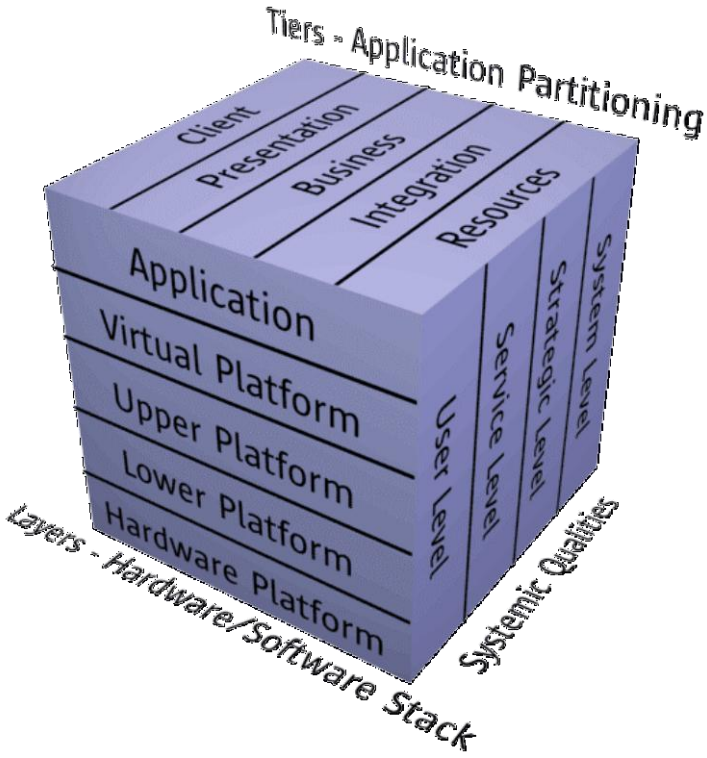

Figura 2.1: Sun Architectural Methodology

2.1.1 Conceito de Tier

As *tiers* oferecem uma organização lógica ou física dos componentes numa cadeia ordenada de consumidores e fornecedores de serviços. Os componentes de uma dada *tier* consomem serviços da *tier* adjacente situada a montante e fornecem serviços a uma ou mais *tiers* adjacentes situadas a jusante. Existem cinco *tiers* que se encontram descritas mais pormenorizadamente de seguida:

- *Client Tier* - Esta *tier* representa todos os dispositivos ou sistemas que o cliente pode utilizar para aceder a um sistema ou aplicação. Um cliente pode faze-lo usando um *browser*, um PDA, uma Java applet, um telemóvel ou outro qualquer dispositivo ou sistema que tenha esse propósito.
- *Presentation Tier* - Esta *tier* encapsula toda a lógica aplicacional necessária para apresentar o serviço ao cliente. Intercepta os pedidos do cliente e é responsável pela gestão da sessão, acede a *business tier*, constrói a resposta e entrega-a ao cliente.

- *Business Tier* - Esta *tier* fornece todos os serviços requeridos pela aplicação e pelo cliente. Contém a lógica e os dados dos serviços prestados. Todo o processamento da aplicação está centralizado nesta *tier*.
- *Integration Tier* - Esta *tier* é responsável pela comunicação com os recursos ou sistemas externos. A *Business Tier* usa esta *tier* sempre que seja necessário obter um objecto que resida na *Resource tier*.
- *Resource Tier* - Esta *tier* contém todas as fontes de informação e como tal os dados, que podem ser internos ou externos aos sistema.

2.1.2 Conceito de Layer

As *layers* são usadas para descrever a tecnologia que os serviços usam em cada uma das *tier*, as plataformas de *software*, *hardware* e de rede utilizadas que permitem que um componente disponibilize um serviço. Enquanto as *tiers* representam as cadeias de processamento nos componentes as *layers* representam as relações do componente durante a implementação e a instalação dos serviços. Existem cinco *layers* que se encontram descritas de seguida:

- *Application Layer* – Contém todos os componentes específicos de uma aplicação. Alguns exemplos são: páginas HTML ou JSP, Sevlets, classes EJB, etc.
- *Virtual Platform Layer* – Esta *layer* fornece ferramentas para desenvolvimento de aplicações empresariais.
- *Upper Platform Layer* - Esta *layer* fornece todos os serviços e interfaces que são necessários a camada acima. Consiste em produtos de *middleware* como servidores web, servidores aplicacionais, sistemas de gestão de bases de dados, etc.
- *Lower Platform Layer* - Desta *layer* fazem parte todos os serviços fundamentais tais como correio electrónico, serviço de nomes (DNS) e outros que são acessíveis através de um conjunto de protocolos específicos.
- *Hardware Layer* - Todos os componentes físicos que suportam a infra-estrutura de *software* do sistema fazem parte desta *layer*.

2.1.3 Systemic Qualities

As *Systemic Qualities* descrevem as estratégias, as ferramentas e as práticas que são utilizadas para atingir uma determinada qualidade de serviço. A lista seguinte descreve algumas das *systemic qualities* mais importantes (3).

- *Usabiliy* - reflecte a facilidade com que os utilizadores conseguem executar as tarefas
- *Performance* - tempo que é necessário esperar até que uma acção seja completada
- *Reliability* - mede quantas vezes o sistema falha ou o tempo entre falhas
- *Availability* - percentagem de tempo que o sistema está disponível para ser usado
- *Accessibility* - incorpora paradigmas de usabilidade para aqueles que tem limitações físicas
- *Manageability* - capacidade de conseguir facilmente iniciar, reiniciar e parar o sistema ou alguns dos seus processos, de o supervisionar de modo a conseguir o comportamento correcto e de tomar medidas correctivas caso seja necessário.
- *Security* - capacidade de impedir o acesso ao sistema e aos dados por parte de quem não tem acesso a este.
- *Serviceability* - é o grau de facilidade que um sistema pode ter para ser reparado ou actualizado através da substituição de cada componente.
- *Scalability* - é a capacidade de conseguir suportar mais utilizadores sem ser necessário alterar ou redesenhar a arquitectura
- *Flexibility* - é a facilidade de adicionar novos serviços à arquitectura e de melhorar os já existentes
- *Reusability* - capacidade de componentes de um sistema serem incorporados noutro
- *Portability* - capacidade de um componente migrar para outra plataforma sem alterações significativas

2.2 Concretização: Arquitectura BicNet

A arquitectura genérica seguida neste projecto segue o modelo da arquitectura tridimensional explicada anteriormente (3). Este modelo de uma arquitectura distribuída e escalável é descrita de seguida discriminando *tiers*, *layers* e *systemic qualities*.

2.2.1 BicNet Tiers

Na Figura 2.2 podemos ver os componentes da arquitectura BicNet (3) (4) e as dependências entre eles. As ligações representam as dependências entre os componentes a nível de funcionalidades fornecidas. Esta arquitectura é baseada em Java/J2EE no entanto os canais de comunicação com os componentes de mais baixo nível podem não ser implementados em Java.

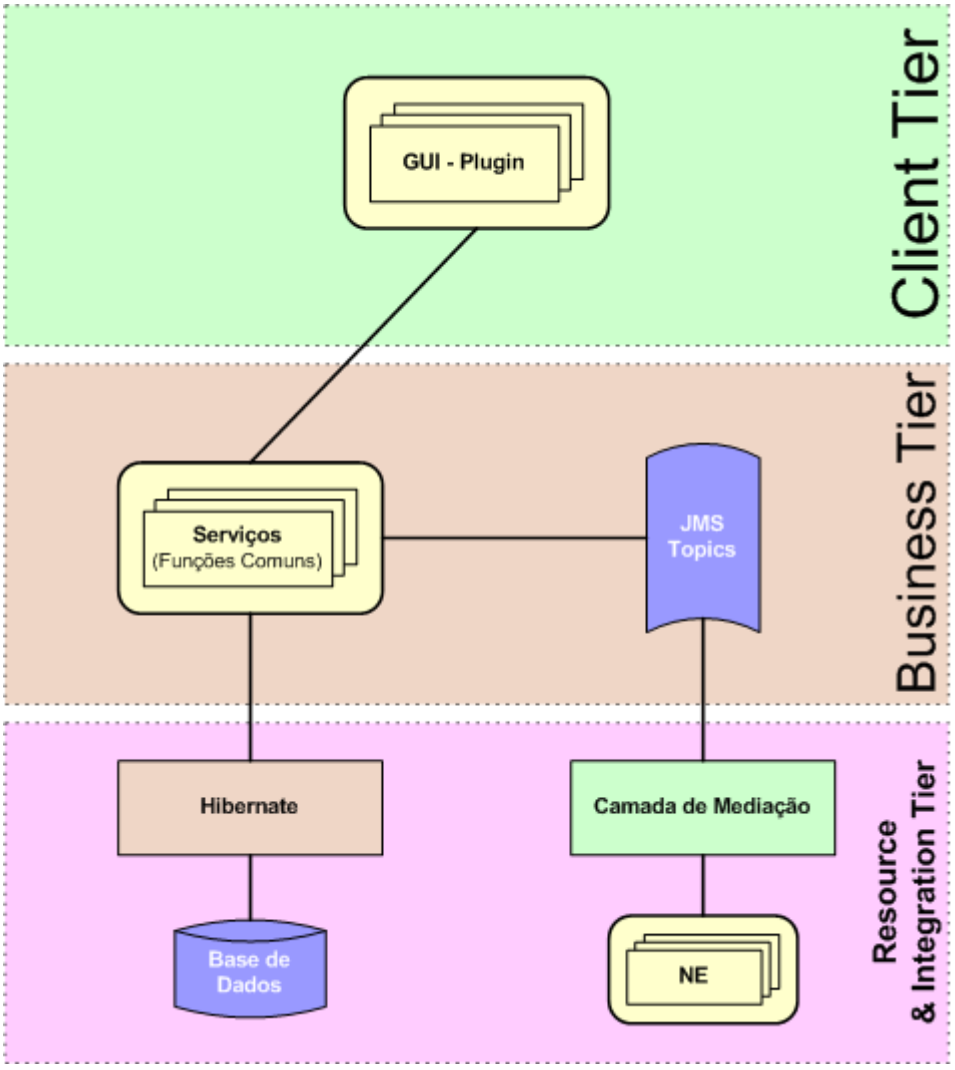

Figura 2.2: Arquitectura Bicnet

Na figura 2.2 estão representadas três *tiers* que em larga medida seguem o modelo de arquitectura já apresentada e que englobam:

- Client Tier - Esta *tier* contém todos os clientes que acedem aos serviços disponibilizados. Os clientes podem ser, por exemplo, os Gui-Plugins que fornecem uma interface gráfica que facilita a interacção com os serviços que são disponibilizados pelo produto (5).

- Bussiness Tier - Esta *tier* contém a lógica da aplicação. Engloba um conjunto serviços que são disponibilizados por componentes denominados Funções Comuns. Estes foram separados de acordo com os requisitos funcionais. Cada Função Comum possui e é responsável por gerir de um modo independente o seu conjunto de objectos. No entanto existe um modelo de objectos comuns que é partilhado por todas. Este modelo partilhado permite prevenir cópias de objectos, manter o modelo de dado coerente e sincronizado entre todas as Funções Comuns e diminuir os efeitos negativos que um modelo não optimizado poderia ter a nível de desempenho e escalabilidade. Devido ao facto de o modelo ser partilhado irá haver um aumento na dependência e na comunicação entre estas Funções Comuns. O modelo é o BicNet Communication Bus (BCB) (6) que define interfaces públicas para representar os objectos geridos. Cada Função Comum implementa as interfaces dos objectos geridos que necessita podendo sempre fazer extensões ao modelo de dados. É através do BCB que cada Função Comum pode aceder a objectos que são geridos por outras Funções Comuns.

- Integration Tier - Esta *tier* contém todos os componentes que são responsáveis por mapear objectos do modelo de dados BicNet para o respectivo recurso. Estes componentes podem ser um mapeador objecto-relacional como o Hibernate no caso de haver a necessidade de persistir os objectos, ou uma camada de mediação no caso de serem dados que têm de ser sempre lidos a cada acesso e não necessitarem de ser persistidos.

- Resource Tier - Esta *tier* engloba todos os recursos que contêm os objectos geridos pelo sistema ou seja a Base de Dados e todos os componentes abaixo da camada de mediação ou seja os componentes nativos e físicos do sistema.

2.2.2 BicNet Layers

A implementação de uma aplicação com a complexidade e a dimensão de uma plataforma deste tipo acarreta algumas decisões que influenciam a estabilidade e o futuro da mesma. Isto significa que é necessário utilizar produtos que são fiáveis e têm qualidades que são desejáveis neste sistema.

Estes sistemas têm de ser capazes de escalar de um cenário muito pequeno com poucas máquinas para uma rede com centenas de utilizadores concorrentes.

As aplicações com várias *tiers* têm um nível de complexidade muito grande e requerem integrar recursos, sistemas e componentes diferentes. A plataforma J2EE já provou ser um padrão para a implementação deste tipo de sistemas devido às suas características modulares e oferta de uma grande quantidade de serviços e APIs. Esta plataforma oferece um conjunto de recursos para aplicações empresariais como Gestores de Bases de Dados, Monitores de Transacções, Serviço de Nomes, etc. O J2EE usa uma aproximação orientada aos componentes que é uma aproximação desejável neste tipo de sistema.

O Jboss foi escolhido como servidor aplicacional. O Jboss é um servidor aplicacional *open-source* baseado em J2EE e implementado em Java. Este utiliza um tipo de arquitectura modular que é atingido através do uso de Java Management Extensions (JMX). O Jboss na sua versão actual suporta quase todas as funcionalidades oferecidas pelo J2EE 6.0.

No modelo BicNet é utilizado um gestor de Base de Dados (DBMS do inglês Database Management System) de modo a persistir a representação da rede gerida. Este Gestor pode ser diferente dependendo do produto ao qual este modelo é aplicado. A implementação do modelo BicNet baseado numa plataforma J2EE deve ser o mais independente possível do DBMS.

2.2.3 BicNet Systemic Qualities

É necessário que a arquitectura BicNet tenha um conjunto de características que são desejáveis de modo a que esta possa incutir as qualidades necessárias aos sistemas onde será aplicada. Talvez a mais desejada destas características seja a escalabilidade. É necessário que os sistemas escalem rapidamente vertical e horizontalmente. Esta característica pode ser garantida, numa solução agregada, simplesmente adicionando mais máquinas.

A alta disponibilidade é garantida mantendo a redundância necessária no sistema e removendo os pontos únicos de falha. Esta característica pode ser garantida se for utilizada também uma aproximação usando agregados (*clusters).*

A segurança é também um ponto-chave. Para resolver este ponto existe um componente específico na arquitectura que trata de todos os mecanismos relativos à matéria. O componente é o Security Manager. Este é responsável pela autenticação de utilizadores, segurança na transferência de objectos, comunicação segura, bloqueio à intrusão de agentes estranhos ao sistema e todas as outras funções que se prendem com a segurança do mesmo.

Para um sistema já implementado é importante que possa ser estendido para novas versões de componentes, mais recentes, rapidamente e sem um risco elevado para a integridade do mesmo. Isto pode ser atingido separando os vários componentes da aplicação em blocos, de modo que ao haver uma alteração, apenas os blocos afectados sejam substituídos. É também desejável que a arquitectura do sistema já contemple mecanismos de tolerância a falhas, recuperação e migração.

A manutenção do código tem de ser simples. Para tal é necessário que se sigam um conjunto de "*Best Practices*". A arquitectura BicNet reduz a complexidade ao separar o sistema em *tiers*, ao usar *Design Patterns* e princípios de desenho orientado a objectos, focando na encapsulação e separação dos componentes. Nesta arquitectura são usados componentes externos que simplificam o trabalho de manutenção como o uso do Hibernate (7) para persistência.

2.2.4 Funções Comuns BicNet

As Funções Comuns são componentes genéricos que podem ser reutilizados transversalmente em vários produtos de gestão. As Funções Comuns são definidas na linguagem Java usando a plataforma J2EE. O ciclo de vida, a instalação e o deploy são independentes para cada uma (7).

Uma Função Comum é constituída por um ou mais dos seguintes componentes: Cliente (GUI Plug-in), Servidor Lógico, Modulo Comum (Objecto com uma função bem definida - ex: Objectos J2EE) e um repositório persistente de informação (3).

A arquitectura genérica de uma Função Comum encontra-se representada na figura 2.3.

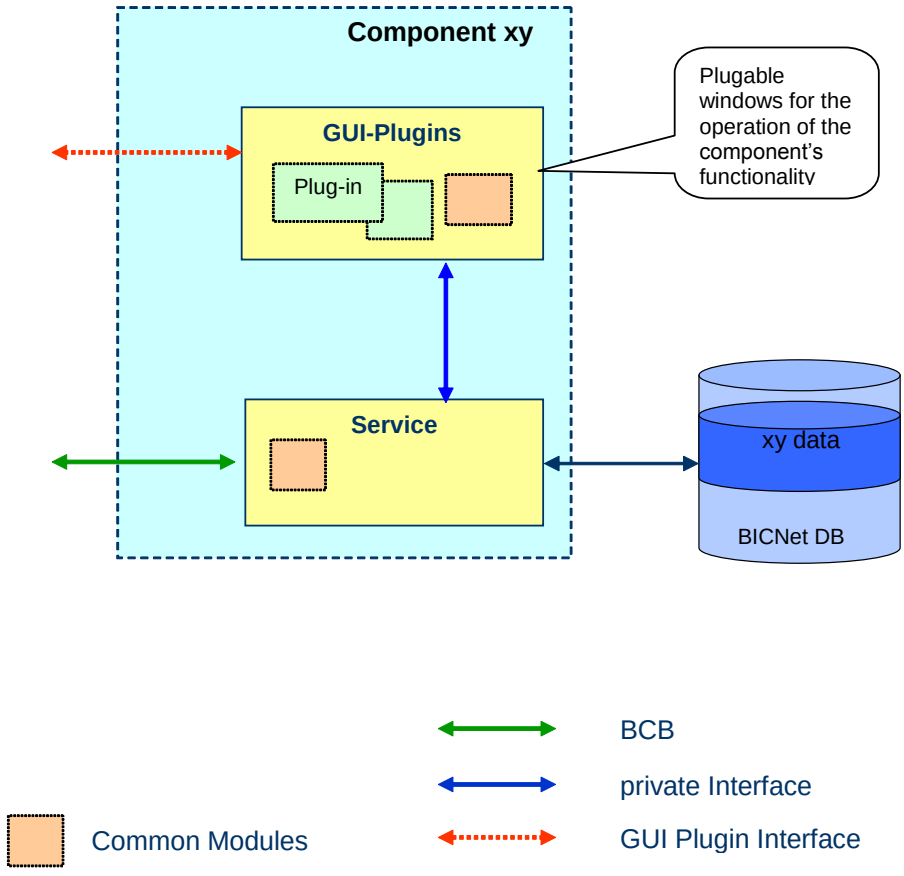

Figura 2.3: Arquitectura genérica de uma Função Comum

Toda a comunicação entre as Funções Comuns é baseada num modelo de informação comum BicNet Communication Bus (BCB).

O BCB não é uma infra-estrutura de comunicação mas sim um conjunto de interfaces bem definidas (6). O BCB surge da aplicação do conceito de Model Driven Architecture (MDA) que é a solução da OMG para ajudar a resolver problemas de integração. Este tipo de arquitectura segue uma aproximação onde há uma separação forte e clara entre a especificação das funcionalides do sistema e a implementação dessas funcionalidades. Para atingir este fim a MDA define uma arquitectura para modelos que fornecem um conjunto de linhas condutoras para estruturar especificações de modelos. Deste modo é possível que o mesmo modelo que especifica um dado conjunto de funcionalidades possa ser implementado em várias plataformas do mesmo modo que permite que a integração de diferentes aplicações através da relação directa entre os modelos possibilitando a integração e a interoperabilidade e a evolução do sistema. O primeiro passo quando se constroi uma aplicação baseada em MDA é criar um modelo aplicacional independente da plataforma realizado em UML. Este será o *core model*. Posteriomente este modelo geral será convertido para servir uma plataforma específica.

Cada Função Comum pode definir as suas próprias extensões privadas ao modelo de informação partilhado BCB.

Apesar das Funções Comuns utilizarem o mesmo modelo de informação (BCB) cada uma gere e possui o seu próprio conjunto de objectos. Apenas o modelo de informação é partilhado.

Cada Função Comum tem um conjunto tabelas que representam os objectos geridos pelas mesmas. As tabelas de um componente podem ter um conjunto de chaves estrangeiras para tabelas de outros componentes. No entanto cada Função Comum possui um conjunto de tabelas que representa o seu domínio. Cada Função Comum é responsável por gerir as tabelas que possui ou seja as que fazem parte do seu domínio.

O acesso de uma Função Comum ao seu repositório persistente de informação pode ser feito usando Hibernate ou acesso directo através de JDBC, dependendo do tipo de objecto que se quer aceder.

Existem várias Funções Comuns, cada uma com uma função bem definida:

- User and Security Management - responsável por todos os mecanismos que se prendem com a segurança da aplicação.

- Log Management - componente que é usado por qualquer outro componente BicNet quando necessita de escrever eventos num registo histórico.
- Fault Management - responsável por gerir os alarmes e os contadores de alarmes do sistema.
- EM/NE Object Management - componente que é responsável por administrar os Element Management (EM) e gerir os Network Elements (NE) bem como a relação hierárquica entre eles. Os EMs são objectos que agregam NEs.
- Topology Management – é o componente responsável pela definição e visualização da rede física

2.3 Sumário

Neste capítulo apresentou-se o modelo base da arquitectura tridimensional (*Sun Architectural Methodology*) e descreveu-se o modo como este modelo se encontra concretizado nos produtos de telecomunicações Siemens através da arquitectura BicNet.

Deu-se especial relevo à arquitectura e constituição das Funções Comuns, componentes genéricos de utilização transversal em vários produtos de gestão, incluindo a Gestão Topológica de Redes.

Capítulo 3

Gestão Topológica de Redes

O Gestor da Topologia de Rede (do Inglês Topology Management) é a Função Comum que suporta todas as funcionalidades necessárias para a definição e visualização dos dados relativos à descrição da rede física (9) (10). Disponibiliza também esses dados a outras aplicações *End-to-End* (E2E) para que estas possam construir automaticamente a sua própria vista sobre a rede gerida. Os dados disponibilizados são:

- Physical Trails (PT) - Ligação física entre dois portos - Physical Termination Points (PTP's) .
- Network View - Este objecto pode ser disponibilizado pelo Topology Management ou por outro servidor E2E e representa uma vista sobre a rede gerida. O Topology Management mantém a vista da rede que representa a configuração da rede física – Physical Network View. Outras aplicações E2E podem manter outras vistas da rede – Logical Network View. Estas vistas lógicas da rede podem conter mais informação específica sobre a rede ou um segmento desta. As aplicações E2E podem criar uma vista temporária da rede contendo apenas os objectos afectos a um recurso de rede. As várias vistas da rede podem ser apresentadas aos vários clientes ao mesmo tempo. Uma Network View pode conter Topological Containers, Topological Symbols, Graphical Links e Graphical Route Elements. As Network Views estão acessíveis através de uma interface e podem ser implementadas pelos componentes que a desejarem disponibilizar.
- Topological Symbol (TS) - Um Network Element é representado graficamente no mapa através de um Topological Symbol. Este objecto disponibiliza algumas informações sobre o NE que representa tais como alarmes, estado, etc.

- Topological Container (TC) - Permite a definição da estrutura da rede de um modo similar a um sistema de directorias. Um TC pode conter TS's e outros TC's.
- Graphical Link (GL) - Objecto que representa todas as Physical Trail entre dois NEs. Um Graphical Link pode representar uma ou mais Physical Trails.
- Graphical Route Element - Permite mostrar as rotas no mapa de rede.

3.1 Enquadramento

Como a Função Comum Topology Management é um componente que faz parte do modelo BicNet, depende e assume a existência de: um Servidor aplicacional J2EE, do modelo de informação dos serviços BicNet – BicNet Communication Bus (BCB), das funcionalidades da @P Platform e dos serviços de nomes e notificações (8).

A @P Platform é uma camada da qual fazem parte componentes que oferecem serviços genéricos às Funções Comuns como pode ser visto na figura 3.1 (11).

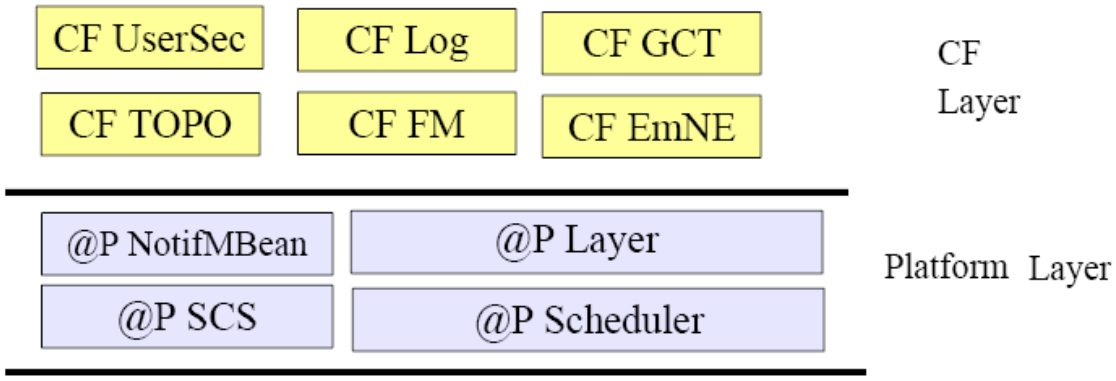

Figura 3.1: @P Platform

Todos os componentes, Funções Comuns ou outros clientes, quando necessitam de retirar informações sobre os NEs irão apenas chamar a funcionalidade que o @P Layer disponibiliza através de uma API genérica. A @P Layer faz parte da camada de mediação na arquitectura da Figura 2.2 como pode ser observado na Figura 3.2.

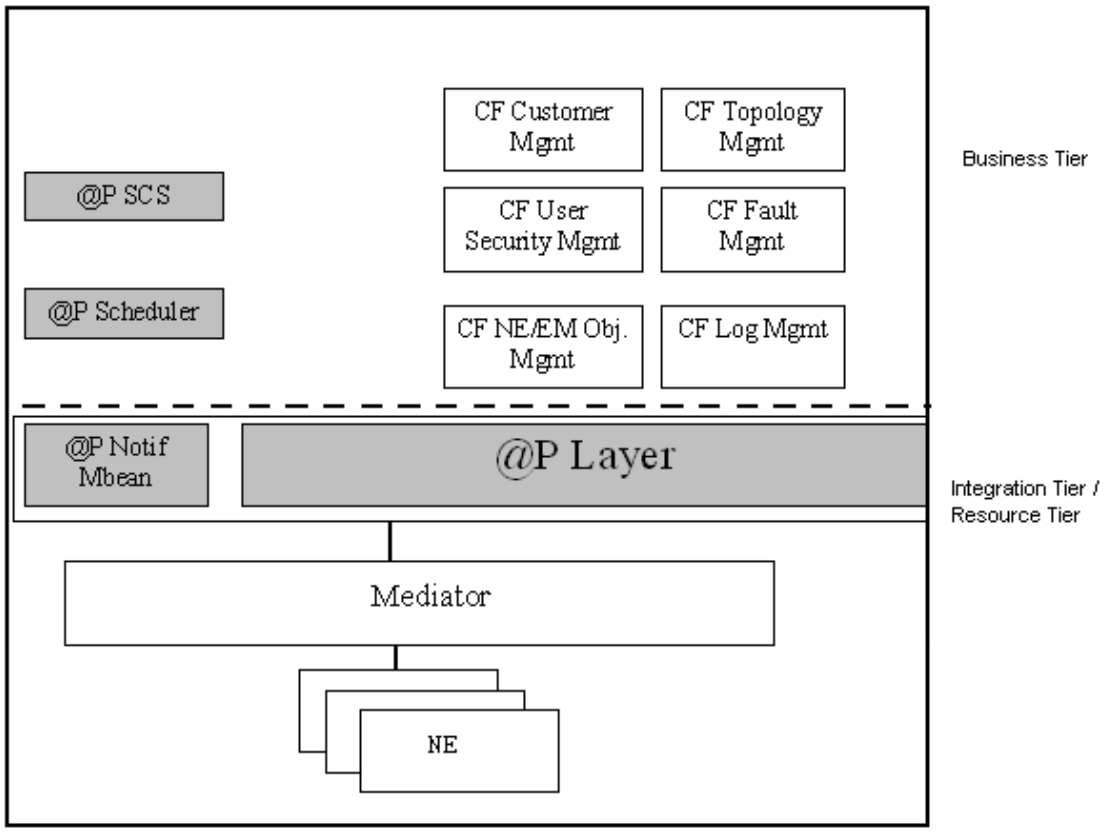

Figura 3.2: @P Layer

Deste modo não existe a preocupação por parte de quem acede a essa funcionalidade de conhecer as especificidades de acesso aos NEs. Se não houvesse este papel da @P Layer todas as outras Funções Comuns teriam de aceder directamente aos NEs para retirarem informações sobre estes. Isto provocaria a replicação do código de acesso e iria interferir com os princípios básicos de desenho como encapsular e separar conceitos.

São enviadas notificações através do @P Notif Mbean quando um elemento da rede muda de estado ou tem um alarme novo. O @P Notif Mbean é o componente responsável pela gestão das notificações do sistema. É o componente ao qual se delegam as notificações que necessitam de ser publicadas nos tópicos JMS (12) correspondentes. Este processo pode ser observado na Figura 3.3.

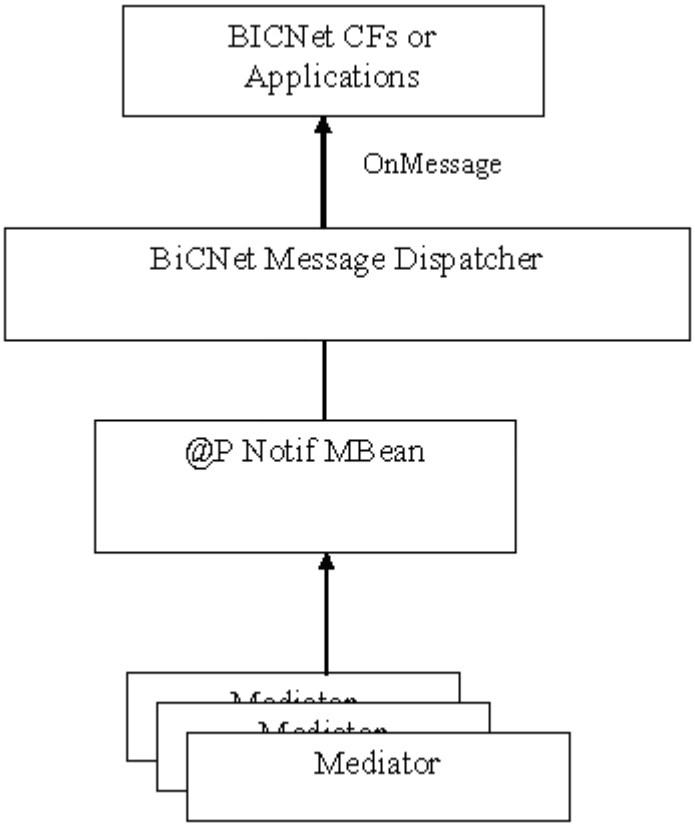

Figura 3.3: @P Notif MBean

A @P Layer e o @P Notif Mbean já referidos fazem ambos parte da @P Platform. Outros componente

s como o @P SCS responsável, entre outras, pela sincronização de todos os componentes e o @P Scheduler responsável pelo agendamento de tarefas também estão incluídos na @P Platform.

3.2 Arquitectura

A arquitectura genérica de qualquer Função Comum já foi apresentada. No entanto cada Função Comum pode interagir com diferentes componentes BicNet, dependendo da informação que necessita para executar as suas funcionalidades. A Figura 3.1 mostra a arquitectura da Função Comum Topology Management e as suas interacções com outros componentes BicNet.

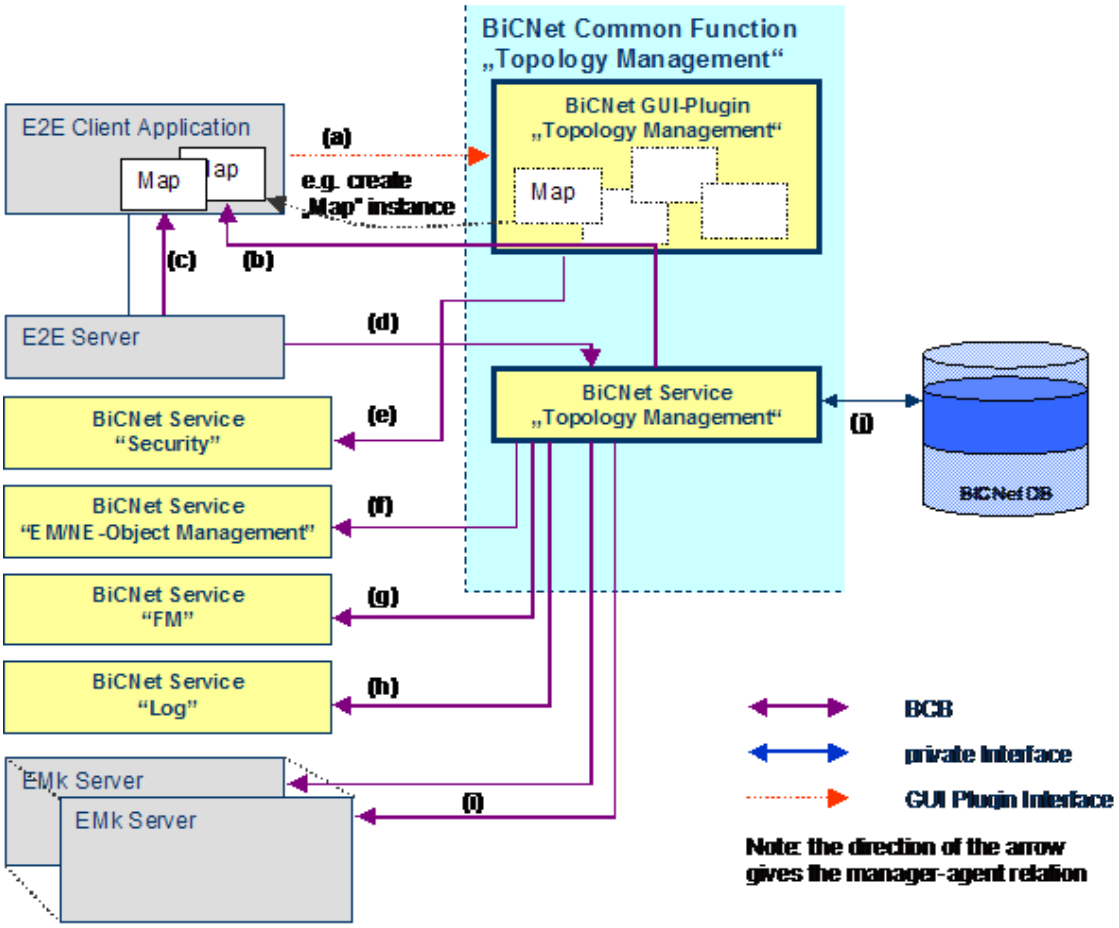

Figura 3.4: Arquitectura da Common Function Topology Management

Observando a Figura 3.4 verificamos que como qualquer Função Comum, o Topology Management consiste num BicNet Service “Topology Management”, a parte de servidor que não tem parte gráfica e um GUI Plug-in “Topology Management”, a parte cliente que fornece um ambiente gráfico onde é possível efectuar funções de gestão (10). A comunicação entre estes dois componentes é necessária por forma a permitir a construção e actualização da vista sobre a rede. A informação contida nessa vista é persistida do lado do BicNetService - servidor. O Gui-Plugin, para apresentar graficamente a vista sobre a rede gerida, pede a informação ao BicNetService. Quando há uma alteração na vista executada no Gui-Plugin é enviada uma actualização ao BicNetService de modo a essa alteração ser persistida. A comunicação dá-se nos dois sentidos uma vez que as actualizações não são sempre despoletadas no Gui-Plugin.

Podem chegar notificações ao BicNetService que provocam o envio de uma actualização para o Gui-Plugin.

A apresentação da vista sobre a rede pode ser feita por outra aplicação E2E. Esta interacção do Topology Management com outras aplicações E2E está representada na Figura 3.4 com as letras (d) e (b). A aplicação E2E comunica com o BicNetService Topology Management para recolher os dados necessários a construção da sua própria rede. Quando o Topology Management Gui-Plugin é utilizado do contexto de uma aplicação E2E este comunica com o servidor E2E em vez de contactar o BicNetService Topology Management.

O Topology Management Gui-Pugin pode ser invocado a partir de dois contextos.: no contexto do Topology Management ou no contexto de outra qualquer aplicação E2E. Quando este Gui-Plugin é chamado no contexto do Topology Management todos os NEs e PTs são apresentadas. Esta vista está disponível para ser apresentada mesmo quando não há aplicações E2E. Quando o Gui-Plugin é invocado usando uma aplicação E2E que usa uma determinada tecnologia apenas os NEs que suportam essa tecnologia são apresentados. Para que o servidor E2E possa contactar o Gui-Plugin este necessita de disponibilizar e implementar as interfaces do BicNetService Topology Management para o Gui-Plugin.

O Topology Management depende ainda de algumas Funções Comuns (10):

- *Security Manager (Security)* - Para permitir a configuração e a execução de políticas de segurança para que seja possível, por exemplo, aceder ou esconder objectos, consoante o tipo de utilizador. A interacção com este componente está representada na Figura 3.4 pela letra (e).
- *EM/NE (EM/NE - Object Management)* – Para permitir receber a lista de todos os NEs conhecidos e notificações relativas a mudança de estado dos mesmos. A interacção com este componente está representada na Figura 3.4 pela letra (f).
- *Fault Management(FM)* - Para receber notificações de alarmes provenientes dos vários objectos geridos e actualizar os contadores de alarmes para esses objectos. Na Figura 3.4 podemos observar esta relação, representada pela letra (g).

- *Log Management (Log)* - Para registar selectivamente as operações para as quais esse registo está especificado. Observando a Figura 3.4 verificamos que esta interacção se encontra representada pela letra (h).
- *Element Management (Emk server)* - A comunicação com este componente é feita através da @P Layer. Este componente disponibiliza toda a informação relativa aos NEs e aos PTPs. Na Figura 3.4 podemos observar esta relação representada pela letra (i).

Na Figura 3.4 podemos observar que existe uma ligação, representada pela letra (j), entre o Servidor e a Base de Dados. Como qualquer Função Comum, o Topology Management requer um repositório persistente de informação. Este é necessário para o armazenamento de informação que necessita de mantida entre sessões, neste caso, utiliza-se uma Base de Dados. A comunicação é feita usando Hibernate. Por questões de desempenho e de gestão todas as interacções com a base de dados devem ser geridas pelo Hibernate.

3.3 Comunicação

A comunicação pode ser discriminada a três níveis: entre a camada cliente e a do servidor ou seja entre o Gui-Plugin e um serviço, no servidor entre dois serviços e entre o servidor e a camada que contém os recursos de rede. De recordar que qualquer Função Comum se enquadra na arquitectura BicNet, já apresentada, em que se consideravam (Figura 2.2) as *tiers*: *client*; *business*; *integration/resource*.

3.3.1 Comunicação Client Tier – Business Tier

O acesso às funcionalidades disponibilizadas pelas Funções Comuns por parte de um cliente é feito através de um conjunto de *Session Beans* que definem uma interface remota e uma interface local. Existe pelo menos uma instância de um *Stateless Session Bean* (SLSB) por Função Comum que contém todos os métodos fornecidos por esta ao "mundo" como se pode observar na Figura 2.2. As funcionalidades fornecidas por cada Função Comum são implementadas em POJOs, aos quais os pedidos recebidos são delegados, de forma a realizar a lógica do mesmo. Estas classes Java implementam a interface do SLSB e tem acesso aos objectos geridos pela Função Comum através do *Domain Object Model* (DOM). As funcionalidades fornecidas podem ser requisitadas

por qualquer cliente. Pode haver a necessidade no entanto de haver comunicação com um cliente específico. Neste caso pode ser disponibilizado somente para esse (o Gui-Plugin respectivo de cada Função Comum), um conjunto de interfaces privadas às quais só ele tem acesso (3) (5). O processo está detalhado na Figura 3.5.

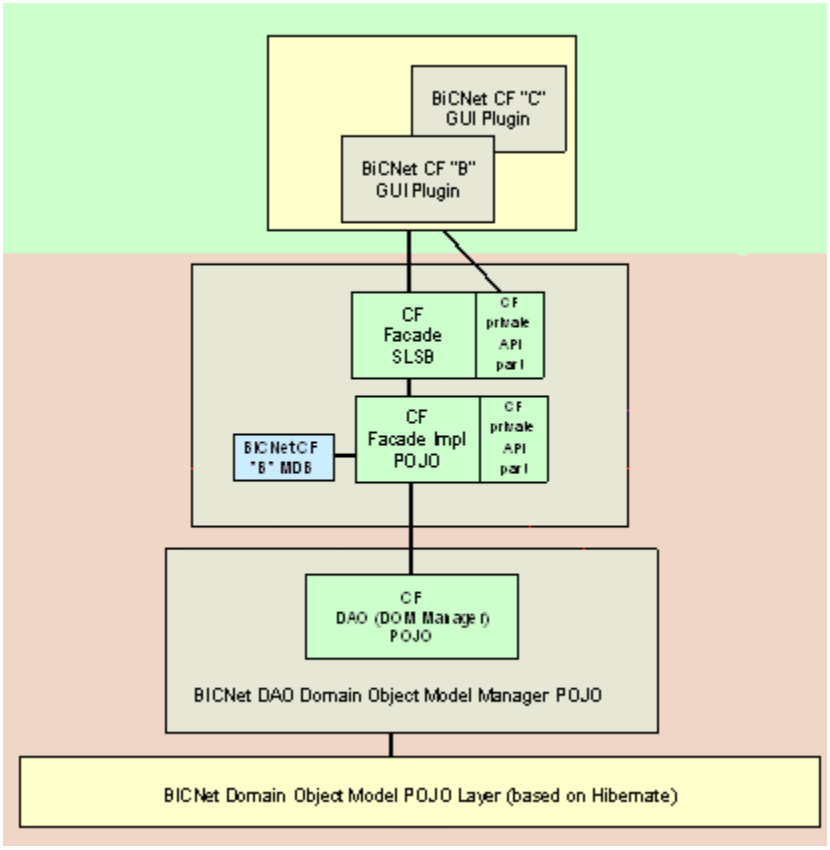

Figura 3.5: Comunicação client tier – business tier

O *BicNet Domain Object Model* pode ser dividido em duas camadas: a camada de *Data Acess Objects* (DAOs) e a camada de objectos DOM. Esta última representa a rede física de objectos geridos enquanto os objectos DAO consistem em classes que implementam as funcionalidades básicas de *create*, *retrive* ou *get*, *update* e *delete*. Todas as Common Function tem um conjunto de DAOs que permitem executar as acções acima referidas sobre os objectos por elas geridos.

Para invocar um método da interface pública do BicNet CF SLSB o Gui-Plugin tem de primeiro obter a referência remota da instância do SLSB que implementa a interface (POJO). Esta referência vai ser adquirida através de um *Service Locator*. No servidor a chamada ao método é delegada a classe Java que implementa a interface do *Stateless Session Bean*.

3.3.2 Comunicação Business Tier – Integration/Resource Tier

A informação retirada da *resource tier* pode ser: informação da Base de Dados ou informação sobre os NEs. A informação contida em Base de Dados é acedida usando Hibernate enquanto que a informação dos NEs é retirada usando a camada de mediação (Figura 2.2). De seguida estão descritos os dois processos.

Camada de Mediação

Quando as Funções Comuns necessitam de aceder a informações sobre os NEs podem requisita-las através da @P Layer. Esta última disponibiliza vários métodos ao "mundo" que permitem a recolha da informação sobre um ou um conjunto de NEs. A Função Comum invoca o método da @P Layer que lhe permite retirar a informação que necessita. Esta é responsável por esconder os mecanismos de acesso ao NE e retornar os dados pedidos. Neste caso é feita uma chamada síncrona.

Nem sempre a comunicação é síncrona, a comunicação assíncrona pode ser realizada através de um sistema de notificações. Quando uma notificação é levantada por um NE, esta é trazida até a @P Layer. Esta última está registada como fornecedora de notificações. Quando a notificação chega à @P Layer é publicada na infra-estrutura de comunicação BicNet, que é gerida pelo componente @P Notif Mbean. Para isto é necessário fazer uma chamada à interface do SLSB disponibilizado pelo @P Notif MBean de modo a retirar a referência para a interface. A referência é retirada através de um *Service Locator*. Com essa referência é criada uma instância do @P Notif MBean SLSB e é chamado o método apropriado para a mensagem que acabou de ser recebida. O @P Notif MBean SLSB vai então delegar a mensagem à classe que implementa o método que foi chamado, de modo a realizar a lógica da notificação. Quando uma nova mensagem é publicada, todos os consumidores de mensagens que sobrescreveram esse tópico, irão ser informados através da invocação dos métodos onMessage() para cada um. Este processo pode ser observado na Figura 3.3. No caso de o sobrescritor ser uma implementação *server-side* o método onMessage() faz parte de um *Message Driven Bus* (MDB) que delega o processamento da mensagem ao método associado que faz a implementação da interface do objecto (3).

Hibernate

O Hibernate é um mapeador objecto-relacional que permite persistir informação numa Base de Dados e aceder a esta de uma forma muito rápida. Esta ferramenta permite a definição de conceitos que existem no Java tais como herança, polimorfismo, associações entre outros. O Hibernate suporta os maiores sistemas de gestão de Bases de Dados tais como Oracle, Microsoft SQL Server, MySQL, SAP DB, DB2, etc.

Existem tipos básicos no Hibernate que depois são mapeados para os tipos SQL que existem no modelo de Base de Dados. Estes tipos básicos são: *integer, long, short, float, double, caracter, boolean, byte, string, date* e *timestamp* (7).

Todos os tipos básicos permitem o valor NULL. Além dos tipos básicos podemos também ter colecções. Estas não são consideradas tipos básicos uma vez que não mantém o seu estado na tabela da entidade que as contém.

As grandes vantagens de usar o Hibernate são :

- Rapidez na execução de interrogações (querys)
- Possibilidade de configurar transacções para interagirem com um Servidor Aplicacional
- Desenvolver objectos persistentes de uma maneira muito similar ao Java sendo possível usar conceitos como polimorfismo, herança, associação, composição e a framework de colecções Java.
- Uso de uma linguagem própria (HQL) que contém poucas alterações ao SQL e que fornece uma ponte elegante entre os objectos e o relacional

O Hibernate não tenta implementar o seu próprio gestor de transacções ou a sua própria *cache* partilhada *multithread*. Toda a parte transaccional é delegada na ligação à Base de Dados. O Hibernate pode ser visto como um adaptador muito leve à volta do JBDC que adiciona uma semântica orientada a objectos.

É necessário, no entanto, ter alguns cuidados quando se criam sessões Hibernate. Nunca se deve criar mais que uma sessão por ligação a Base de Dados, porque as sessões guardam todas as alterações que se vão fazendo à base de dados para elas próprias, por isso, diferentes sessões podem ver valores não actualizados. Se duas

threads necessitarem de usar a mesma sessão é necessário existir um mecanismo externo de sincronização, ou seja, as sessões não são *threadsafe*.

O Hibernate pode ser utilizado com um modelo de *lock* optimista de modo a libertar o programador de algum trabalho ao gerir versões e a enviar excepções sempre que se tenta aceder a dados inválidos. No entanto para assegurar a atomicidade das transacções algumas regras têm de ser tidas em conta (3):

- Fase 1 – com o *auto-commit* ligado obter uma sessão e chamar um dos métodos *find* ou *load* para trabalhar com os objectos que são necessários
- Fase 2 – fazer as alterações desejadas. Depois não fazer nenhum *find* ou *commit* pois isto iria fazer *flush* a sessão
- Fase 3 – desligar o *auto-commit* (i.e iniciar a transacção). Isto permite fazer tudo o que é necessário com a sessão e acabar com *commit* ou *cancel*

O Hibernate consegue persistir colecções ou um conjunto de objectos como java.util.Map, java.util.Set, java.util.List e arrays de objectos. No entanto tudo o que é adicionado pela classe que implementa a interface não é suportado como por exemplo a ordenação no *TreeSet*. Só é suportado o que está definido na interface. Isto porque o Hibernate substitui as instâncias de *Map*, *Set* e *List* pelas suas próprias instâncias persistentes destas colecções.

As colecções podem ser inicializadas *lazily*. Isto significa que o estado destas só é carregado quando a aplicação pede uma destas colecções. Esta funcionalidade é de extrema importância quando são usadas muitas colecções. Esta inicialização lenta é transparente e é também a razão principal para o Hibernate ter as suas próprias implementações de colecções.

O diagrama da figura 3.6 mostra a integração das camadas de servidor com as camadas de dados persistentes:

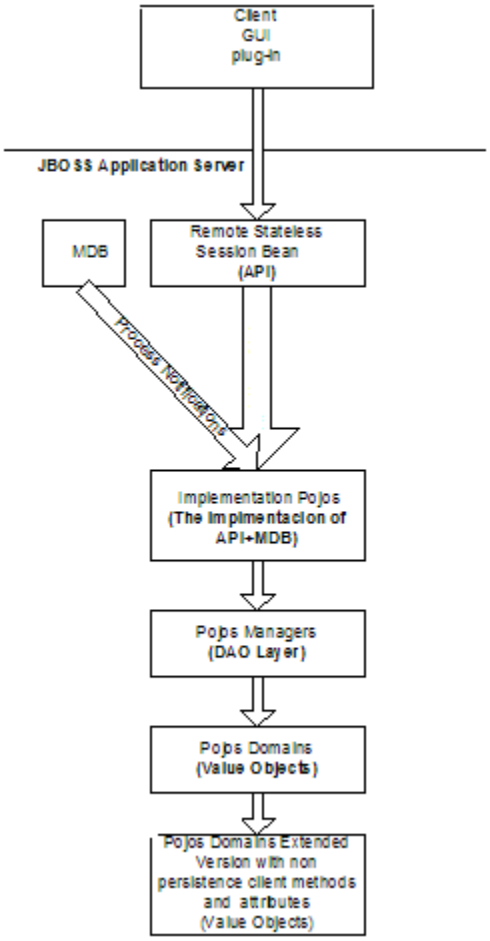

Figura 3.6: Camadas do Servidor Aplicacional

O servidor aplicacional tem cinco camadas :

- A API do servidor - tem dois modos de entrada: síncrono e assíncrono. No modo síncrono os métodos dos *Stateless Remote Session Beans* são chamados sincronamente. No modo assíncrono é via notificação processada pelos *Message Driven Beans*.
- POJOs - onde reside 90% do código das Funções Comuns BicNet.
- POJO Managers - Esta é a camada de persistência o servidor aplicacional implementa o CRUD (Create, Retrieve, Update e Delete). O Hibernate Syncronizer vai automaticamente gerar esta layers ou seja os DAOs.
- POJOs Domains – estas classes são classes que reflectem uma linha de uma dada tabela na Base de Dados. Esta camada também pode ser gerada automaticamente pelo *Hibernate Syncronizer*.

- Extensão do POJOs Domains – esta camada estende os atributos e os métodos *get* e *set* da camada POJOs Domains apenas para satisfazer necessidades específicas das aplicações cliente. Os campos que não são persistidos mas que apenas façam sentido nas aplicações cliente devem ser adicionados a esta camada. Alguns exemplos destes campos podem ser cores, permissões, activação/desactivação de acções, etc. Estes campos são perdidos quando o cliente se desliga.

3.3.3 Comunicação Business Tier – Business Tier

As Funções Comuns podem necessitar de apresentar informação que é mantida por outra Função Comum. É necessário que elas comuniquem para poderem recolher as informações que necessitam. A comunicação entre Funções Comuns dentro da *business tier* é toda feita utilizando o modelo de informação comum BicNet Communication Bus (BCB). Tal como acontece na comunicação entre a *client tier* e a *business tier*, o acesso as funcionalidades disponibilizadas pelas Funções Comuns por parte de outras Funções Comuns é feito através de um conjunto de *Session Beans* que definem um interface remota e uma interface local. Existe pelo menos uma instância de um *Stateless Session Bean* (SLSB) por Função Comum que contém todos os métodos fornecidos por esta ao "mundo". Cada Função Comum disponibiliza um conjunto de métodos que são conhecidos e podem ser acedidos pelo "mundo".

3.4 Serviços Disponibilizados

O Topology Management fornece um conjunto abrangente de funcionalidades relacionadas com a gestão e supervisão da rede. Essas funcionalidades encontram-se descritas de seguida (10):

- Definição da Rede: A rede física pode ser definida através do Topology Management, através da criação de TCs que permitem estruturar a rede da maneira que o utilizador deseja, de TSs e PTs.
- Supervisão da rede: Para atingir este objectivo o Topology Management tem de interagir com outras Funções Comuns. Cada Função Comum disponibiliza um tipo de informação. A informação relativa a alarmes dos NEs é fornecida pelo *Fault Management* (cf. Figura 3.4). Cada TS no

*Topology Managemen*t tem um ícone associado com a cor da severidade mais alta do alarme que do NE possui. Existem várias cores para os alarmes e várias severidades sendo que a mais alta é *Critical* com a cor vermelha e a mais baixa é *None* com a cor verde. É possível através deste componente ver a listagem de todos os alarmes afectos a um NE ou a uma PT. Os alarmes das PT são o somatório de todos os alarmes dos seus NEs. Os TSs apresentam também informação relativa ao estado do NE, se está activado ou desactivado, se está a iniciar ou a encerrar, se ocorreu algum erro na inicialização, etc. Esta informação é fornecida pela Função Comum EMNE e é uma mais-valia na supervisão da rede.

- Mapa da Rede: é utilizado pelo *Topology Management* para apresentar graficamente a rede física. Este mapa de rede pode ser usado também por outras aplicações E2E.
- Mecanismos de segurança suportados pelo Topology GUI Plug-in: os mecanismos de segurança são suportados pela chamada a um módulo comum de segurança. Esse módulo comum de segurança é também uma Função Comum – Security Management (cf. Figura 3.4).

3.4.1 Apresentação da Rede

A forma de apresentação da rede física pode ser feita de dois modos: *Topological Map* (mapa) ou *Topological Tree* (árvore). Através do *Topological Map* e da *Topological Tree* é possível definir e supervisionar a rede. A definição da rede é feita adicionando ao mapa ou a árvore os elementos que compõem a rede e a supervisão é atingida através da recolha de informação pelo componente responsável pela gestão da topologia de rede para apresentação (9).

Quando a forma de apresentação é em árvore, como se ilustra na figura 3.7, é apresentada uma vista hierarquizada de todos os *Topological Containers* e *Topological Symbols*. A posição geográfica dos mesmos não é considerada e as *Pysical Trails* não são mostradas neste tipo de apresentação contrariamente ao que acontece quando a forma de apresentação é o mapa. O mapa quando é aberto está sempre afecto a um *Topological Container*. Quando o mapa é aberto inicialmente o *Topological Container* que lhe está afecto será o *Container* que contém todos os restantes elementos da rede. Posteriormente podemos navegar entre *Topological Containers* e ampliar certas regiões

do mapa. Para ilustrar estes cenários em baixo encontram-se modelos de uma *Topological Tree* (cf. Figura 3.7) e de um *Topological Map* (cf. Figura 3.8) (10).

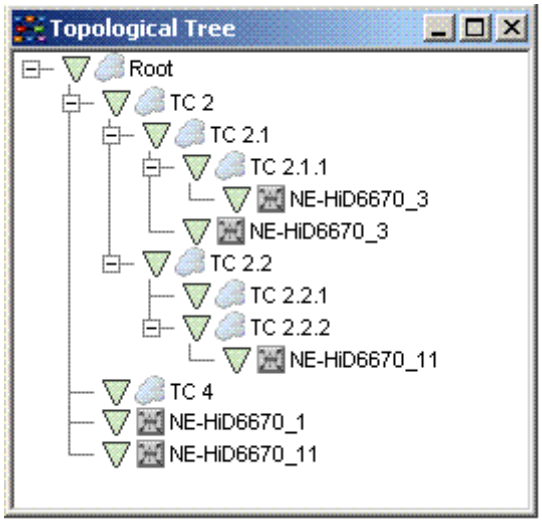

Figura 3.7: Topological Tree

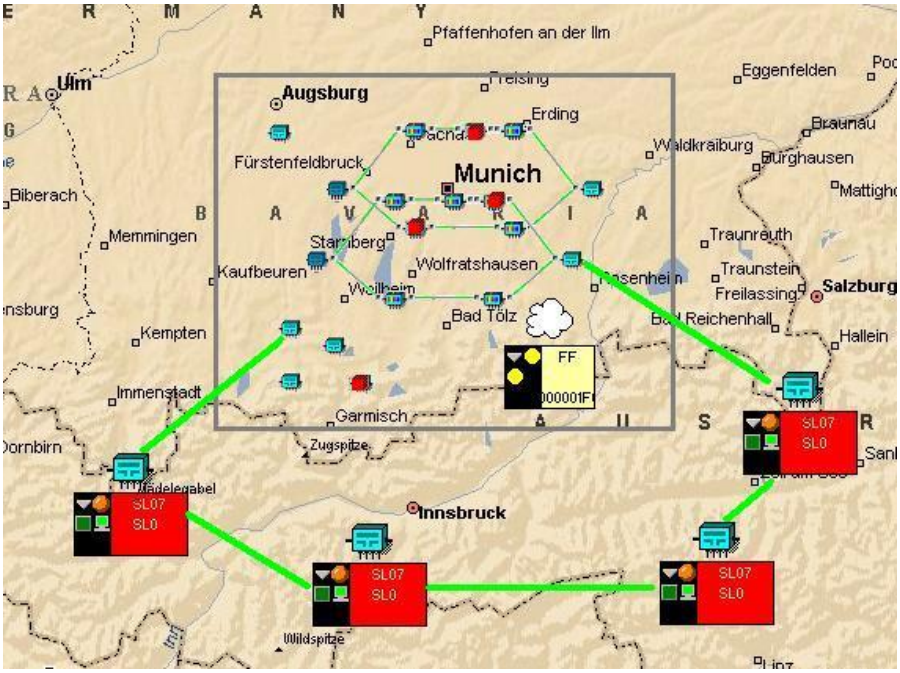

Figura 3.8: Topological Map

O *Topological Map* é uma GUI disponibilizado pelo *Topology Management* que apresenta a *Network View*. Através do mapa podemos mudar a configuração da rede, criando, modificando ou apagando os objectos que são apresentados. Podemos também organizar a rede de um modo semelhante a um sistema de directórios utilizando TCs. Cada TC pode conter outros TCs ou TSs. Deste modo podemos melhor organizar a rede

física por exemplo usando os TCs para representar cidades, regiões e agrupar os TSs por TC. Cada TS pode ter um ou mais ascendentes enquanto cada TC apenas tem um.

O *Topological Map* necessita da implementação da interface *Network View* para poder ser apresentado. Esta interface permite não só aceder aos objectos mas também suportar as notificações recebidas aquando da criação, alteração ou remoção de objectos. A *Network View* representa uma vista sobre a rede e pode conter *Topological Symbols*, *Graphical Links*, *Topological Containers* e *Graphical Route Elements*. Esta vista pode ser disponibilizada pelo *Topology Management* ou por outra aplicação E2E que implemente as interfaces necessárias à disponibilização da mesma (3).

O conceito de filiação é de extrema importância para o *Topology Management*, pois influencia a organização e configuração da rede. O *Topological Map* é sempre aberto através de um TC. O TC através do qual mapa é aberto é o *Active Container*. Em cada *Active Container* apenas são visíveis: os seus descendentes directos ou seja os TCs e os TSs dos quais ele é ascendente e os seus descendentes indirectos (TSs com o ícone mais pequeno e TCs representados com o ícone de uma nuvem, não deixando deste modo visualizar o seu conteúdo). Para poder visualizar o resto da rede é necessário continuar a navegar no mapa.

A *Topológical Tree* é outra forma de apresentação da *Network View* disponibilizada pelo *Topology Management*. Nesta a apresentação é feita de forma hierárquica e não são apresentados os GLs. A árvore tem vários nós, cada nó da árvore representa um TC ou um TS. Nesta representação da *Network View* apenas a hierarquia lógica é considerada e não a posição dos TSs ou TCs. Devido a este facto, se existir um TS que seja descendente do TC x e também do TC y, na árvore, este símbolo irá aparecer repetido por baixo do TC x e do TC y. Observando a Figura 3.7 podemos verificar esta situação. O NE NE-HID6670_11 encontra-se repetido como descendente dos *Topological Containers* Root e TC2.2.2.

O modo de pedir os dados do servidor nos dois modelos de representação da rede gerida é *lazy*, ou seja, os dados só são pedidos quando é preciso que sejam apresentados, no caso da árvore quando se expande um nó ou no caso do mapa quando se abre um TC.

Desempenho na Apresentação de Dados

A questão do desempenho é de extrema importância num componente como o *Topology Management* pois este reúne muita informação de outros componentes para posterior apresentação.

Um problema grave de solução não trivial deste componente era o tempo de espera para visualização da rede. Uma rede com quinhentos TSs demorava um minuto a ser desenhada. Por questões de desempenho este componente já dispunha de uma *cache* de ícones. Os TSs são representados com ícones como pode ser visto na Figura 3.8. Os ícones apresentados não fazem parte do conjunto de objectos do *Topology Management*. Quem tem acesso a esses objectos é a Função Comum EMNE. O *Topology Management* pede esses ícones ao EMNE e constrói uma cache onde guarda os ícones de modo a não necessitar de os pedir novamente. Esta melhoria apesar de significativa não foi suficiente.

Foi necessário investigar, tarefa essa que constitui um desafio interessante, onde se encontravam os estrangulamentos da aplicação. Sempre que o mapa físico é aberto é necessário fazer uma chamada ao servidor a pedir todos os *Topological Symbols*, *Topological Containers* e *Graphical Links* existentes na rede. Para cada um dos símbolos é ainda necessário pedir algumas informações a outros componentes uma vez que para estes objectos é disponibilizada muita informação de controlo ao operador.

Estas chamadas aos outros componentes apesar de necessárias eram responsáveis pela demora na visualização do mapa.

A solução para o problema consiste em realizar uma *cache* dos atributos pedidos de modo a não ser sempre necessário fazer esses pedidos. A tabela *Topological Symbol* incorpora campos novos. Os que seriam pedidos aos outros componentes nomeadamente EMNE, *Fault Management* e *Element Management*. Estes campos contêm informação que o TS irá apresentar na visualização da rede. Uma vez que o modelo de dados foi alterado toda a estrutura que suporta o acesso ao mesmo teve de ser também alterado. Além dessa alteração também o suporte a actualização destes atributos terá de ser fornecida.

Quando o símbolo é criado as informações relativas aos novos campos são pedidas a cada um dos componentes que gere essa informação. A informação é então colocada em Base de Dados. A informação pode ser modificada, por exemplo, o nome do NE (um dos atributos incorporado na tabela *Topological Symbol*) pode ser alterado.

Quando uma alteração é realizada ou um atributo muda de estado essa informação é notificada. Para os componentes saberem da alteração basta serem consumidores dessa notificação. Sempre que uma notificação de uma alteração de algum atributo novo mantido na tabela *Topological Symbol* é recebida, o estado do atributo é actualizado na Base de dados e é enviada uma notificação para o Gui-Plugin para o estado ser também actualizado no cliente.

A actualização pode ainda ser feita por outro mecanismo. Periodicamente ou sempre que há uma perda de notificações, o componente responsável pela gestão das mesmas despoleta uma sincronização de todos os componentes. A sincronização é feita pelo componente @P SCS, que pede a todos os componentes que se sincronizem. Cada um dos componentes é responsável pela sua própria sincronização. Cada componente sabe o que precisa para ficar sincronizado. Nestes casos específicos de manutenção dos novos atributos, sempre que há uma sincronização são pedidas de novo as informações aos componentes que gerem estes novos atributos. As informações vindas dos outros componentes são persistidas na Base de Dados e é enviada uma notificação ao Gui-Plugin para actualizar o estado de um modo semelhante ao que é feito na criação. Deste modo é assegurado que os atributos novos colocados na tabela *Topological Symbol* estão sempre actualizados.

3.4.2 Gestão de Links

O *Topology Management* é o componente responsável pela gestão de *links* (*Physical Trails*) entre elementos de rede. É dada, através duma interface gráfica *user-friendy*, a possibilidade ao operador de criar apagar e modificar PTs.

A criação de uma PT no *Topology Management* é feita através da ligação entre dois portos (PTPs) compatíveis como pode ser observado na Figura 3.9.

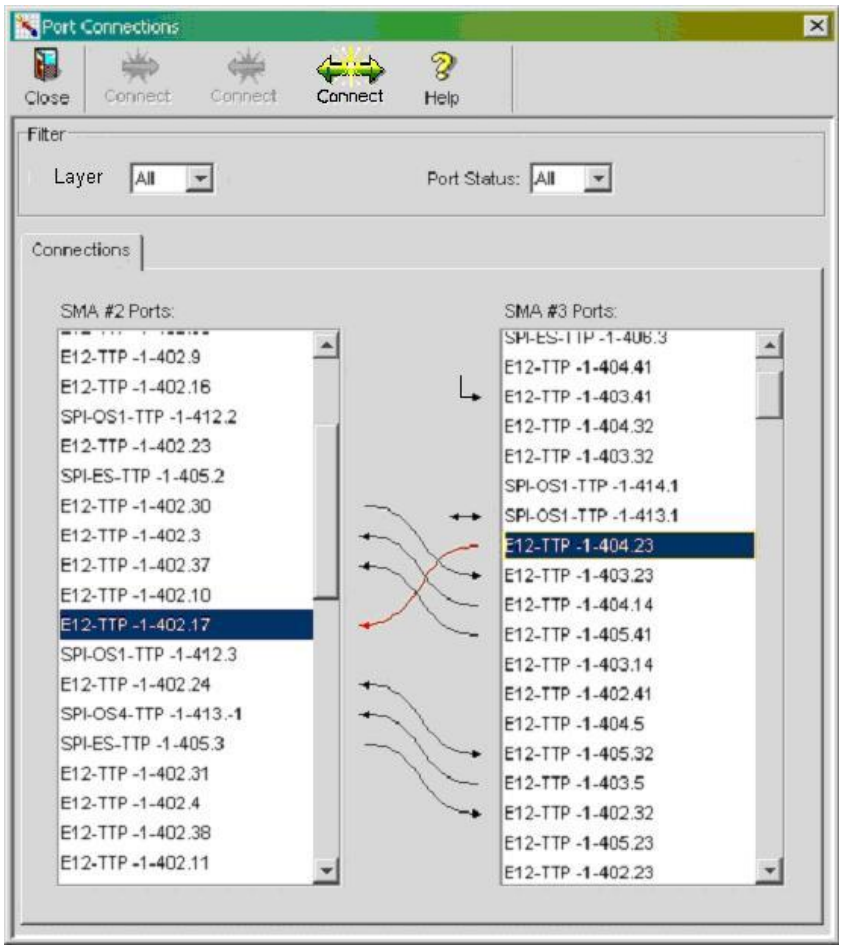

Figura 3.9: Criação de uma *Physical Trail*

Apesar de ser dada a possibilidade de criação de PTs ao operador existem também PTs que vão ser criadas devido a notificações que são enviadas por um processo chamado descoberta automática de *links*. Estas notificações enviadas pela camada de mediação são despoletadas pela descoberta de *links* na rede. A camada de mediação envia a notificação de PT criada que é automaticamente criada no *Topology Management*.

Uma lista de PTs que existem na rede é disponibilizada pelo *Topology Management* e através desta lista é possível supervisionar o estado das mesmas. Sobre estas PTs muita informação é disponibilizada, tal como: o estado operacional (*enable*/*disable*), a direcção (uni ou bidireccional), os portos e os alarmes para cada um destes, o nome, entre outros. É dada a possibilidade de modificar os atributos que cada PT tem assim como de apagá-la. É possível apagar uma PT via *Topology Management* ou esta pode ser apagada devido ao envio pela camada de mediação de uma notificação para apagar uma determinada PT.

Alguns atributos destas PTs podem ser alterados pelo utilizador, no entanto, o estado desta nem sempre é igual. Sempre que há uma alteração das PTs, essa alteração é notificada. Para receber notificações sobre alterações das PTs é necessário ser consumidor dessas notificações, sempre que uma notificação chega é tratada. A alteração é persistida em Base de Dados e é enviada uma actualização ao cliente para que a apresentação gráfica reflicta o estado da PT.

A actualização pode ainda ser feita por outro mecanismo, através da sincronização despoletada pelo componente @P SCS. Este vai pedir a todos os componentes que se sincronizem. Cada um dos componentes é responsável pela sua própria sincronização. Quando uma sincronização é despoletada é forçada a actualização das PTs. A informação que é retirada durante a sincronização é persistida e enviada uma actualização ao cliente para que a apresentação gráfica reflicta o estado da PT.

No *Network Map* a representação das PTs é aglomerada num objecto – *Graphical Link* – que representa todas as PTs entre dois NEs.

3.4.3 Detecção Automática de Links

O *Automatic Link Detection* (ALD) é uma nova funcionalidade adicionado ao produto TNMS-DX na sua versão três. O desenvolvimento desta nova funcionalidade foi um desafio para a equipa do *Topology Management*. O ALD consiste na descoberta de PTs existentes e na actualização automática da vista da topologia da rede de acordo com os resultados. Deixa de ser necessária a intervenção do operador na criação das PTs (13).

Algoritmo de detecção de PT

A solução para esta nova funcionalidade ser suportada é complexa. Para o ALD ser suportado é o usado um identificador único na rede, definido para cada porto. A ideia básica consiste em encontrar portos (*source* e *sink*) que contêm os mesmos identificadores. Sempre que são encontrados dois portos com identificadores iguais uma PT é detectada (13).

De modo ao ALD ser suportado este identificador tem de ser único em toda a rede onde o NE está localizado. Quando pedido, o NE irá gerar e garantir a unicidade de um identificador.

O *automaticTraceMode* é um atributo que é usado para activar ou desactivar a geração automática de identificadores únicos. Quando este atributo está activo o NE gera um valor único e guarda-o.

No processo de descoberta de PTs os portos *sink* e *source* irão ser comparados e se forem iguais então existe uma ligação entre eles.

Depois da análise da rede física, as ligações físicas detectadas são comparadas com aquelas que já existiam anteriormente e estão mantidas em Base de Dados. Quando existem diferenças entre a rede física detectada e a rede lógica mantida em base de dados, a rede física a que foi descoberta prevalece. Todas as PTs que foram detectadas e não existem em Base de Dados são criadas assim como todas as PTs que existem em Base de Dados e não foram detectadas são apagadas. Em Base de Dados é mantida a estrutura actual da rede física.

O ALD pode ser despoletado de duas maneiras, manualmente pelo utilizador ou através do agendamento de uma tarefa que irá posteriormente ser executada. No caso de o ALD ser manual o operador só necessita de escolher a opção *Manual* ALD. No caso de o ALD ser agendado o operador terá de escolher a opção *Scheduled* ALD e posteriormente terá de definir se a execução é periódica, semanal ou mensal. Em qualquer um dos três casos o operador terá de definir as especificidades da execução. No caso de a execução ser periódica este terá definir a data e hora de início e a periodicidade da mesma. Se a execução for semanal terá de definir a data e hora de início e o dia da semana que pretende que seja executado o ALD. Para a execução mensal terá de definir a data e hora de início e o ou os dias do mês nos quais deve ser executada a tarefa. De seguida podemos ver três figuras dos três tipos de agendamento.

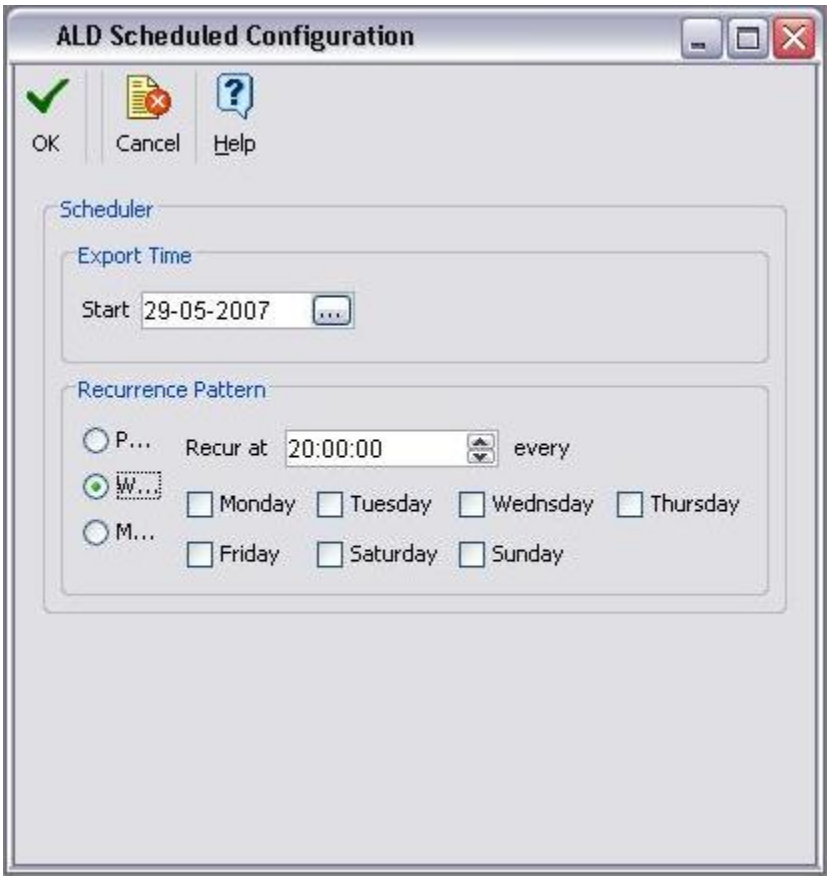

Figura 3.10: Agendamento semanal

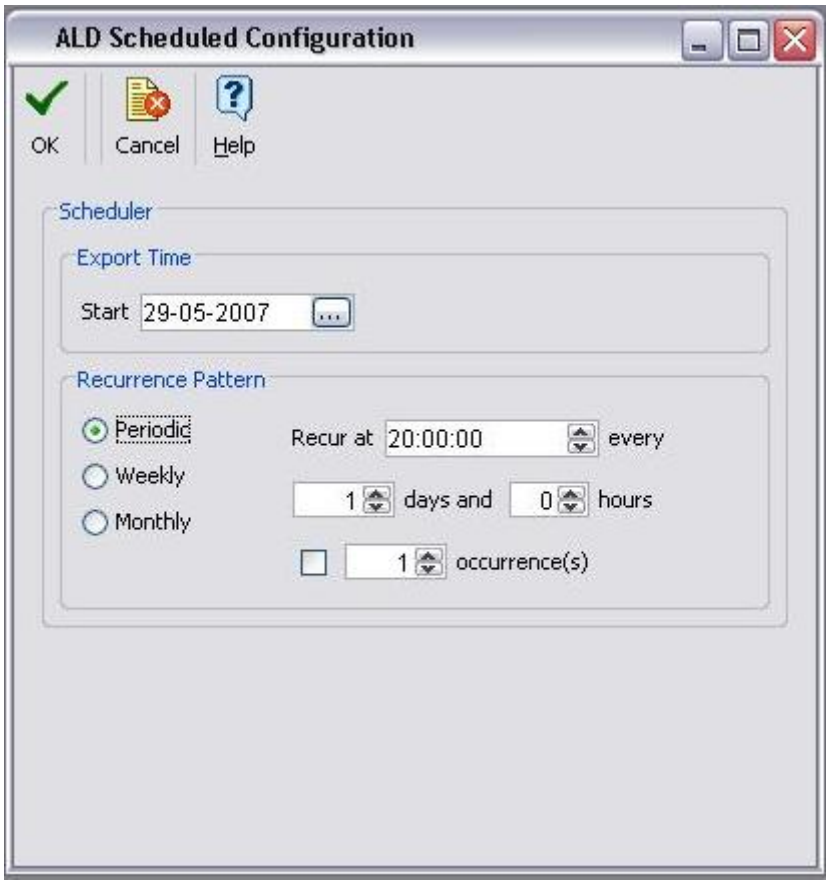

Figura 3.11: Agendamento periodico

Figura 3.12: Agendamento mensal

O agendamento das tarefas está a cargo de uma Função Comum, @P Scheduler. Este é responsável por guardar as tarefas que as outras Funções Comuns agendam e de as despoletar no momento que for definido pelas mesmas (11). Este componente disponibiliza um conjunto de métodos que permitem que outros componentes possam aceder, criar e posteriormente alterar as tarefas.

O *Topology Management* disponibiliza a interface gráfica para definir o agendamento, guarda e valida os dados. Posteriormente cria uma tarefa no @P Scheduler com esses dados. Apenas uma tarefa de ALD pode ser definida. O utilizador não pode agendar mais que uma tarefa de ALD de cada vez. Na altura para que estiver agendada a execução da tarefa o @P *Scheduler* irá executar os procedimentos necessários para a despoletar. Esta tarefa pode ser cancelada e também alterada pelo utilizador usando a interface gráfica disponibilizada pelo *Topology Management*.

Através do algoritmo já apresentado, todos os PTs entre os NEs que existam na rede irão ser detectados, criado e persistidos. Para todos os PTs que existiam antes do ALD começar é verificado se ainda existem. Caso não existam terão de ser apagados

pois já não reflectem o estado da rede actual. No final da execução do ALD a vista do *Topology Management* sobre a rede tem de reflectir o estado actual da mesma.

Casos de Uso

1. Detecção de Physical Trail Info e resolução de Conflitos

Use Case Name	Detecção de *Physical Trail Info* e resolução de Conflitos
Summary	Execução do ALD despoletada pelo Operador ou por um *Schedule*
Actors	Operador ou *Scheduler*
Component	Servidor do *Topology Management*
Actor input	Seleccionar *Start Manual* ALD ou configurar um *Scheduled* ALD
Actor output	--
GUI Handling	--
Pre-conditions	Alguns NEs estão a correr
Normal Flow	➢ Pedir todos os NEs ➢ Pedir para cada NE os seus portos ➢ Para cada porto activar o atributo *AutomaticTraceMode* ➢ Pedir todos os portos outra vez ➢ Fazer uma lista de todos os portos que tem par. (verificar quais os portos que tem o *sink* igual ao *source*) ➢ Para cada porto desactivar o atributo *AutomaticTraceMode* ➢ Marcar as PT lógicas (já existentes em Base de Dados) para apagar ➢ Marcar as PT detectadas para criação. ➢ Para cada PT detectada verificar se ela já existia anteriormente em Base de Dados. Caso já exista a PT detectada irá ser removida da lista de PTs para criação e a PT já existente em Base de Dados irá ser removida da lista para apagar. ➢ Apagar todas as PTs marcadas para serem apagadas ➢ Criar todas as PTs marcadas para criação
Alternative Flow	Se não há NEs a correr o ALD acaba a escreve uma mensagem no log
Exceptions	Erros na Base de Dados, Erros de comunicação, Erros internos

Minimal Guarantee	--
Success Guarantee	--
Associated Requirements	--

2. Configuração de uma tarefa ALD (Scheduler Configuration)

Use Case Name	Configuração de uma tarefa ALD
Summary	O Operador tem de configurar uma tarefa para que vai correr o ALD
Actors	Operador
Component	Cliente do *Topology Management*
Actor input	Data de Início e definição do tipo de agendamento
Actor output	--
GUI Handling	Janela de configuração de uma tarefa agendada (*Schedule*)
Pre-conditions	O sistema está a correr e não existe nenhum ALD a ser executado
Normal Flow	1) O Operador abre a janela de definição de um *Scheduled* ALD 2) O Operador define os paramentos do *Schedule* 3) O Operador carrega no OK e é criado um *Schedule Job* que irá executar o ALD de acordo com o definido 4) A janela é fechada
Alternative Flow	Se não há NEs a correr o ALD acaba a escreve uma mensagem no log
Exceptions	Erros na Base de Dados, Erros de comunicação, Erros internos
Minimal Guarantee	--
Success Guarantee	--
Associated Requirements	--

3. ALD Manual

Use Case Name	ALD *Manual*
Summary	O Operador executa manualmente um ALD
Actors	Operador
Component	Cliente do *Topology Management*
Actor input	Selecionar *Start Manual* ALD
Actor output	--
GUI Handling	Botão no Menu da Aplicação
Pre-conditions	O sistema está a correr e não existe nenhum ALD a ser executado
Normal Flow	➢ O Operado selecciona a opção *Manual* ALD ➢ O ALD é iniciado
Alternative Flow	Se não há NEs a correr o ALD acaba a escreve uma mensagem no *log*
Exceptions	Erros na Base de Dados, Erros de comunicação, Erros internos
Minimal Guarantee	--
Success Guarantee	--
Associated Requirements	--

Para a concretização desta nova funcionalidade algumas decisões sobre a implementação tiveram de ser tomadas.

O *Topology Management* não guarda informação sobre os NEs que estão na rede. Essa informação é guardada pela Função Comum EMNE. Pode haver NEs configurados no EMNE que não foram adoptados pelo *Topology Management*. O processo de adopção consiste em criar o NE no *Network Map* ou para a *Network Tree*.

Num primeiro passo é necessário pedir ao EMNE todos os NEs configurados que estejam activados. Seguidamente é pedido a cada NE os seus PTPs. Esta informação é pedida à camada de mediação pois é ele que tem informação sobre os portos para cada NE. Esta informação pode ser muito avolumada e traz dois problemas. A quantidade de informação retirada da rede o número de chamadas feitas. Para reduzir o tempo que iria

demorar a retirar de todos os PTPs e a accionar o atributo *automaticTraceMode* para todos os NE essa operação é paralelizada. Seguidamente é iniciado o passo seguinte da descoberta de PTs. O passo seguinte consiste em pedir novamente os PTPs aos NEs que agora já tem o atributo *automaticTraceMode* accionado ou seja já conseguem produzir o identificador único. Desta vez as informações são guardadas. Devido a quantidade de informação esta não poderia ser guardada em memória. Sendo assim são criadas tabelas temporárias na Base de Dados para guardar esta informação. A Base de Dados é usada também para o passo seguinte que consiste em comparar os portos *source* e *sink* e assim descobrir PTs. Isto é feito através de uma *query* SQL que faz a esta comparação e cria uma tabela temporária com o resultado da mesma. Esta última vai guardar a informação sobre as PTs que foram descobertas na rede. No entanto podem já existir algumas PTs em Base de dados que não foram descobertas. É necessário verificar se essas ainda existem na rede física. É feita uma comparação entre as PTs detectadas e as que já existiam em Base de Dados. Esta comparação é também feita através de um duas *querys* SQL, cujo resultado é guardado em duas tabelas temporárias. Uma que contém as PTs que vão ser criadas e a outra que contém as que vão ser apagadas. A opção de fazer todo o processamento a nível da Base de Dados prende-se com a quantidade de informação que é necessário processar. Tanta quantidade de informação necessita de muita memória disponível que nem sempre é uma opção viável. No final do ALD todas as tabelas temporárias criadas durante a execução são apagadas.

3.5 Sumário

Neste capítulo foi apresentada a arquitectura e o modo de comunicação específicos da Função Comum responsável pela gestão topológica da rede. Foram também descritas as funcionalidades deste componente de modo a realçar a importância deste num sistema de gestão de redes de telecomunicações.

Procedeu-se ainda ao enquadramento Função Comum Gestão Topológica de Rede (*Topology Management*) na arquitectura global do Sistema.

Capítulo 4

Conclusões

O trabalho aqui apresentado foi o resultado de um processo de integração na equipa das Funções Comuns e no projecto Topology Management. Processo esse que contou com a leitura de documentação interna da Siemens bem como documentação disponível na Internet sobre tecnologias Java. Este processo de integração e análise do sistema é importante para que se possam cumprir os objectivos propostos.

Os sistemas de gestão das redes de telecomunicações possibilitam a gestão das mesmas de uma forma rápida, eficaz e simples. Deste modo as operadoras podem gerir toda a sua rede através de uma interface *user-friendly* que oferece funcionalidades de configuração, gestão e visualização da rede. Um interface gráfico usável facilita o uso do sistema e diminui o tempo de aprendizagem por parte de quem o utiliza. Deste modo, todas as funcionalidades oferecidas pelo sistema são executadas mais rapidamente e eficientemente, aumentando a produtividade e diminuindo a probabilidade de ocorrência de erros.

O tipo de arquitectura modular orientada aos serviços que aqui foi apresentada traz algumas vantagens:

- Escalabilidade - capacidade de estar preparado para o crescimento do sistema.
- Flexibilidade - facilidade de adicionar novos componentes e serviços à arquitectura já existente ou então simplesmente melhorar os já existentes.
- Reusabilidade - característica que permite incorporar certos componentes noutros sistemas com o mínimo de esforço.
- Portabilidade - torna possível que o sistema possa ser migrado para outras plataformas.
- Interoperabilidade - capacidade dos componentes poderem trabalhar uns com os outros e também com os novos componentes que possam ser adicionados.

SunTone Architecture Methodology

A arquitectura apresentada neste documento BicNet assenta sobre a arquitectura SunTone Architecture Methodology.

A resposta da Sun à necessidade de uma metodologia de desenvolvimento é a SunTone Architecture Methodology, que fornece o melhor de metodologias modernas no desenvolvimento de *software*. Esta aproximação incorpora os princípios de ciclo de vida e desenvolvimento incremental articulado com um ciclo de arquitectura, implementação e administração.

O ciclo de vida do SunTone Architecture Methodology é baseado nas seguintes etapas: Arquitectura, Implementaçao e Administração.

O *workflow* deste tipo de metodologia pode ser resumido nos seguintes passos:

- Análise e Design
- Codificação
- Realização de Testes Unitários
- Integração
- Revisão do Design e da Implementação (Opcional)

A SunTone Architecture Methodology é uma aproximação orientada aos casos-de-uso e ao desenvolvimento. Esta metodologia ajuda a descobrir e corrigir erros relacionado com os requisitos, projectos, tecnologias e usabilidade cedo no ciclo de desenvolvimento, antes que eles causem problemas que podem ter grandes impactos no produto.

Servidor Aplicacional: Jboss

Os servidores aplicacionais J2EE estão a ganhar cada vez mais adeptos devido à grande produtividade proporcionada no desenvolvimento de aplicações empresariais distribuídas e à facilidade no aproveitamento de sistemas legados e bases de dados relacionais em novas aplicações. Estes servidores estão a tornar-se a infra-estrutura da nova geração de aplicações empresariais.

Mapeador objecto-relacional: Hibernate

O Hibernate é um mapeador objecto-relacional de alto desempenho. É uma das soluções mais poderosas no mercado. O Hibernate trata de todo o mapeamento entre as classes e tipos Java e SQL. Fornece funcionalidades que permitem retirar dados e a inquirição sobre os mesmos de uma maneira simples reduzindo significativamente o tempo de desenvolvimento. O objectivo deste tipo de ferramenta é reduzir o esforço do programador no desenvolvimento de tarefas relacionadas com a persistência de dados e eliminar o processamento dos dados usando SQL e JBDC.

As grandes vantagens de usar o Hibernate são :

- Rapidez na execução de interrogações (querys)
- Possibilidade de configurar transacções para interagirem com um Servidor Aplicacional como o JBOSS
- Desenvolver objectos persistentes de uma maneira muito similar ao Java sendo possível usar conceitos como polimorfismo, herança, associação, composição e a *framework* de colecções Java.
- Uso de uma linguagem própria (HQL) que contém poucas alterações ao SQL e que fornece uma ponte elegante entre os objectos e o relacional.

Arquitecturas baseadas em Java

A utilização de uma linguagem como o Java e de uma plataforma como o J2EE permitem atingir algumas das vantagens descritas anteriormente tais como portabilidade e interoperabilidade. A linguagem Java é uma linguagem bem-sucedida e popular porque possui caracteristicas desejadas tais como "correr" em qualquer sistema operativo, ser portável, simples, segura e eficiente. A plataforma J2EE fornece alguns serviços genéricos que facilitam o desenvolvimento de aplicações empresariais. Ela contém bibliotecas desenvolvidas para o acesso a base de dados, RPC, CORBA, etc.

Deste modo não há necessidade de desenvolver este tipo de tarefas uma vez que já são disponibilizados pela plataforma. Devido a estas características a plataforma é utilizada principalmente para o desenvolvimento de aplicações empresariais.

O tipo de arquitectura deste sistema utiliza algumas das funcionalidades oferecidas por esta plataforma. A utilização de "Entreprise Java Beans" (EJB) permite a

distribuição de componentes e a utilização de modelos aplicacionais flexíveis e robustos (14) (15). Esta arquitectura e as tecnologias que lhe estão associadas garantem uma infra-estrutura sólida para qualquer sistema de informação. A escolha do tipo de tecnologia pode determinar algumas das características mais desejáveis no produto.

Bibliografia

1. **AG, Siemens.** PPP:D Product Development for Large Systems. 1999.
2. **Yang, Nan.** *SunTone Architecture Methodology Overview.* 2006.
3. **Team, BicNet SDO.** *Global Specification Implementation Functional Specification Level 1 BicNet Design Specification.* s.l. : Siemens AG, 2006.
4. **Graft, Gunter e Mainwald, Hermann.** *Global Implementation Specification Functional Specification Level 2 BicNet Basic Tecnology.* s.l. : Siemens AG, 2006.
5. **Team, BicNet SDA.** *Graphical User Specification Gui Plugin Architecture.* s.l. : Siemens AG, 2006.
6. **Moschcau, Thomas.** *Global Implementation Specification Functional Specification Level 2 BicNet Interface Specification @P- BCB - BicNet Communication Bus.* s.l. : Siemens AG, 2005.
7. **Pugh, Eric e Gradecki, Joseph D.** *Professional Hibernate.* 2002.
8. **Sousa, João P.** *Apresentação: BicNet Common Fuctions.* s.l. : Siemens SA, 2006.
9. **Rebelo, Sergio e Ferreira, Ana.** *Design Specification Topology Management.* s.l. : Siemens AG, 2006.
10. **Mittermaier, Heiner.** *Global Implementation Specification Functional Specification Level 2 Topology Management.* s.l. : Siemens AG, 2006.
11. **Graft, Gunter.** *@P Platform.* s.l. : Siemens AG, 2004.
12. **Verch, Manuel.** *JMS Asynchronous Communication.* s.l. : Siemens AG, 2004.
13. **Antunes, Nuno, Veiga, Jorge e Reimer, Heiko.** *Functional Specification Level 1 Automatic Link Detection.* s.l. : Siemens AG, 2007.
14. **Roman, Ed, Ambler, Scott e Jewel, Tyler.** *Mastering in Enterprise Java Beans.* 2002.
15. **Marinesscu, Floyd.** *EJB Design Patters.* 2002.
16. **Graft, Gunter e Mainwald, Hermann e Outros.** *Converged ICN Management Systems.* s.l. : Siemens AG, 2006.
17. **Flanagan, David.** *Java in a Nutshell.* 2002.
18. **Soley, Richard e Group, OMG Staff Strategy.** Model Driven Architecture. 2000.
19. **Miller, Joaquin, Mukerji, Jishnu e Outros, e.** Model Driven Architecture. 2001.

Glossário

Cluster - também chamado de *Clustering*, *Cluster* é o nome dado a um sistema montado com mais de um computador, cujo objectivo é fazer com que todo o processamento da aplicação seja distribuído aos computadores, mas de forma que pareça com que eles sejam um computador só. Com isso, é possível realizar processamentos que até então somente computadores de alta performance seriam capazes de fazer.

Common Object Request Broker Architeture - é a arquitectura padrão para estabelecer e simplificar a troca de dados entre sistemas distribuídos heterogéneos. O CORBA actua de modo que componentes de software possam comunicar de forma transparente com o utilizador, mesmo que para isso seja necessário operar com outro software, noutro sistema operativo, com outra ferramenta de desenvolvimento. CORBA é um dos modelos mais populares de objectos distribuídos.

Data Access Object - São objectos que fornecem uma interface comum entre a aplicação e uma ou mais repositório de dados como uma base de dados ou um ficheiro. As vantagens de usar este tipo de objectos são que os objectos que contém a lógica da aplicação não precisam de saber o destino dados que manipulam. Como resultado não é necessário mudar o "core" da aplicação se for preciso mudar onde e como os dados são guardados. No entanto o preço a pagar por colocar esta camada de persistência é aumento da quantidade de código que vai ser executado.

Domain Object Models - Este termo é utilizado para indicar os objectos persistidos que fazem parte do modelo de domínio.

Enterprise Java Beans - são componentes server-side que permitem a construção modular de aplicações encapsulando a lógica das mesmas. As específicações destes fazem parte da plataforma Java estando disponíveis na sua API. Estes beans já lidam com os problemas típicos que tem de ser sempre implementados e re-implementados

pelo programador em todas as aplicações tais como persistência, segurança e integridade transaccional deixando-o assim livre para trabalhar noutras problemáticas.

Java Database Connectivity - é um conjunto de classes que fazem parte da API do JAVA que permitem o acesso independente por parte de aplicações Java a bases de dados.

Java Message Service - é um conjunto de classes que fazem parte da API do JAVA que permite a troca de dados e eventos de uma maneira flexível, segura e assíncrona.

Java Platform Enterprise Edition - é uma plataforma de programação para desenvolver e correr aplicações Java distribuídas com arquitectura multicamandas baseadas em grande parte em componentes de software modulares. Esta plataforma inclui várias API's como JDBC, RMI, e-mails, JMS, web services, XML entre outros e define como coordenada-los. O J2EE também tem especificações para alguns componentes unicamente Java como Enterprise Java Beans, servlets, portlets e Java Server Pages. Isto permite ao programador desenvolver plataformas de aplicações empresariais portáveis e escaláveis.

Jboss - é um servidor de aplicações Java. Os servidores de aplicação permitem o desenvolvimento de aplicações distribuídas multi-camadas. Eles agem como a interface entre os clientes e as bases de dados e os sistemas de informação corporativos (ERPs, sistemas legado, etc.). O objectivo do J2EE é especificar uma plataforma com um modelo de componentes e a infra-estrutura básica (segurança, transacções, acesso a bases de dados, etc.) para o desenvolvimento de aplicações corporativas.

Plain Java Objets - POJO são objectos/classes JAVA que seguem um desenho simplificado em contraposição aos EJBs por exemplo. Um JavaBean é um POJO que segue definições rígidas de estrutura (construtor default sem argumentos e métodos que seguem o padrão de getters e setters para seus atributos).

Remote Method Invocation - permite ao programador criar aplicações Java que podem invocam métodos remotos, ou seja métodos cuja implementação não se encontra disponível localmente.

Simple Network Management Protocol - é um protocolo de gestão de redes TCP/IP que facilita o intercâmbio de informação entre os dispositivos de rede. O SNMP possibilita aos administradores de rede gerir o desempenho da rede, encontrar e resolver problemas de rede e planear o crescimento da mesma. O software de gestão de redes segue o modelo cliente-servidor convencional. Neste sistema são utilizados os termos "Gerente" para a aplicação servidora e "Agente" para a aplicação cliente que corre no dispositivo de rede.

Stateful Session Bean - Usando um SFSB o estado entre cliente e componente EJB é mantido durante uma mesma sessão. Por exemplo nos sites de comércio electrónico os objectos que vão sendo adicionados ao carrinho não podem desaparecer quando se muda de página. Terão de ser mantidos até ao final da sessão. Este tipo de bean deve ser usado se:

- For preciso manter o estado entre cliente e bean numa mesma sessão
- O bean representar a "visão" que o cliente tem da camada de negócio, fornecendo serviços aos clientes e gerindo o workflow de outros enterprise beans.

Stateless Session Bean - Os SLSB não retêm o estado das interacções que mantém com um cliente durante a mesma sessão. Não manter o estado corresponde às variáveis de instância do bean. Por não manter estado, todas as instâncias de um SLSB são indistinguíveis para um cliente. Um contentor de EJBs pode se valer desse facto e criar um pool desses beans para melhorar a performance da aplicação. Genericamente usa-se SLSBs quando:

- O componente executa uma actividade genérica para qualquer cliente
- Apenas um cliente deverá ter acesso ao bean por vez
- O componente não tiver dados persistentes, nem mantiver o estado
- Implementar web service.

Lista de Abreviaturas

@P	@Platform
ALD	Automatic Link Detection
BICNet	Best in Class Network Management
CF	Common Function
CORBA	Common Object Request Broker Architecture
DAO	Data Access Object
DBMS	Database Management System
DOM	Domain Objects Models
E2E	End-to-End
EJB	Enterprise Java Beans
EM	Element Manager
EMk	Converged Element Manager based on ACI framework
EMNE	Element Manager Network Element
EMS	Element Management System
GL	Graphical Link
GUI	Graphical User Interface
IDL	Interface Description Language
J2EE	Java Platform Enterprise Edition
JDBC	Java Database Connectivity
JMS	Java Message Service
JMX	Java Management Extension
MBean	Managed Bean
MDA	Model Driven Architecture
MDB	Message Driven Bean
NE	Network Element
POJO	Plain Java Objects
PT	Physical Trail
PTP	Physical Termination Point
R&D	Research and Development
RMI	Remote Method Invocation

SFSB	Stateful Session Bean
SLSB	Stateless Session Bean
SNMP	Simple Network Management Protocol
TC	Topological Container
TNMS – CORE	Telecommunication Network Management System Core
TNMS – DX	Telecommunication Network Management System Domain Unix
TP	Termination Point
TS	Topological Symbol